CacheInf: Collaborative Edge-Cloud Cache System for Efficient Robotic Visual Model Inference

Abstract

Real-time visual model inference is crucial for various robotic tasks deployed on mobile robots in the field, but limited onboard computation power and limited and unstable wireless network bandwidth of the mobile robot constraint the speed of either local computation or offloading (e.g., to remote GPU servers) of the visual model inference; the prolonged inference time also increases energy consumption for the inference on each input. We observe that visual model inference typically computes on local geometries and consecutive inference inputs often partially share similar local geometries, which provides opportunity to cache and reuse the computation results to both speed up local computation and offloading.

Based on these observations, we propose CacheInf, a collaborative edge-cloud cache system for efficient robotic visual model inference. CacheInf first profiles the visual model and schedules for optimal offloading / local computation plan at different ratio of reusable cache. At runtime, it analyses the incoming inference input and manages reusable cache on both the robot and the server; then CacheInf dispatches local computation and offloading on reduced input size by reusing cached computation results both on the robot and the server, which accelerates both local computation and offloading. Evaluation on various visual models and wireless network environments shows that CacheInf reduced average end-to-end inference latency by TODO% to TODO% and reduced average energy consumption for inference on each input by TODO% to TODO%.

1 Introduction

Visual information is vital for various robotic tasks deployed on real-world edge devices (typically mobile robots), such as surveillance, health care and autonomous vehicles; and as a major visual information processing method, fast visual model inference is important for the real-world robotic tasks to timely respond to environment changes. Unfortunately, the mobile robots often suffer slow visual model inference, because they are typically limited in computation power and have limited and unstable wireless network bandwidth, making both local computation and naively offloading the computation to GPU servers slow.

To tackle this problem, we observe that introducing the classic caching mechanism into robotic visual model inference has the potential to accelerate both local computation and offloading the computation to GPU servers. It is based on the facts that the input for the robotic visual models is typically a continual stream of images and these models mostly compute on the images using local operators (i.e., operators such as convolution that relies on local geometries of the input images). In such cases, computation results of similar local geometries between consecutive images can be reused to shrink the overall size of the computation, which reduce both local computation time and transmission time when offloading computation to the GPU server.

However, the existing caching systems for visual models are designed for interactive generative image edition on high-end PCs and unfit for robotic tasks on mobile robots. With their targeted scenario, they assume no perspective changes in the images, consume too much memory and does not consider the acceleration opportunity of offloading the computation to other GPU servers. A new caching system specifically designed for robotic visual model inference is desired.

To fulfill this gap, in this paper, we propose CacheInf, a collaborative edge-cloud cache system for efficient robotic visual model inference. Given a continuous stream of visual input in a robotic visual task, CacheInf analyses the overlapping area between consecutive inputs; based on the portion of overlapping area (reusable cache) and the current estimated wireless network bandwidth, CacheInf schedules on the action between reusing local cache to reduce local computation time and reusing the remote cache (e.g., the cache at the GPU server side from the robot's perspective) to reduce transmission time, to ultimately reduce the overall visual model inference latency.

The design of CacheInf is non-trivial. The first challenge is to transform cached results to local computation acceleration. While the computation result of cached areas can be reused, the sparse and fragmented remaining un-cached area can not be computed on the highly optimized local operators for dense local geometries, hindering acceleration . To make use of these highly optimized local operators, in CacheInf we search for a minimal set of rectangles that covers the uncached areas and recombine them to a new dense rectangle while preserving local geometries. Computation results on the new rectangle can then be broken up and combined with the cache to form the correct result.

The second challenge is how to reduce the cache memory consumption, especially at the robot side which typically has a tight GPU memory budget. In visual models, the computation results of local operators typically consumes

significantly more memory than the parameters of local operators and naively caching all the computation results leads to heavy GPU memory burden.

We observe that in visual models, the computation result of a local operator often serves as the input of another local operator. In this case, the computation result of the combined rectangle of uncached areas can be directly passed to the following local operators without loss of information, and cache between these two operators can be ignored to reduce memory consumption. We greedily search for a continuous sequence of local operators whose starting and ending operators incur the least memory consumption and cache the computation results of the starting and ending operators only, so as to minimize cache memory consumption.

To fully exploit the potential acceleration of cache and offloading, we also integrate an emerging offloading paradigm named Hybrid-Parallel [8]: during visual model inference on an image, Hybrid-Parallel enables splitting of the input of local operators and assigns different splits to the local robot and the remote GPU server for computation, so that local computation and data transmission of one image can be parallelized to reduce inference latency. We extend the scheduler of Hybrid-Parallel to further consider the potential acceleration with cache, such the splitting and assigning will benefit (TODO) from cache on both the robot and the server.

We implemented CacheInf using python and pytorch on Ubuntu20.04. The offloading of computation is handled by the highly optimized distributed module of pytorch with cuda-aware mpi backend which directly accesses GPU buffer, so as to minimize offloading overhead. Our baselines include plain local computation, a state-of-the-art computation offloading system named TODO, together with its counterpart with cache enabled modified by us, and Hybrid-Parallel.

We evaluated CacheInf on a four-wheel robot equipped with a Jetson NX Xavier that is capable of computing locally with the low power consumption GPU. The offloading GPU server is a PC equipped with an Nvidia 2080ti GPU. Our datasets include the standard datasets of video frames of DAVIS [6] and CAD [1] each captured by a hand held camera and our self captured video frames using sensors on our robot. Extensive evaluation on various visual models and wireless network bandwidth circumstances shows that:

- CacheInf is fast. Among the baselines, CacheInf reduced the end-to-end inference time by XX% to XX%.
- CacheInf saves energy. Among the baselines, CacheInf reduced the average energy consumed to complete inference on one image by XX% to XX%.
- CacheInf is also memory-efficient. The above advantages were obtained by only incurring XX% to XX% increase in memory consumption for CacheInf.

The major contribution of this paper is our new edge-cloud collaborative caching paradigm to accelerate robotic visual model inference, which reuses cached computation results to both accelerate local computation and computation offloading to remote GPU servers. The resulting system, CacheInf, collaboratively considers and reuses cached computation results on both the robot and the server and schedules the computation and offloading to minimize visual model inference latency. The accelerated visual model inference and the reduced power consumption will make real-world robots more performant on various robotic tasks and nurture more visual models to be deployed in real-world robots. The source code and evaluation logs of CacheInf is available at TODO.

The rest of the paper is organized as follows. Chapter two introduces background and related work. Chapter three gives an overview of CacheInf and Chapter four presents its detailed design. Chapter five describes implemented. Chapter six presents our evaluation results and CacheInf seven concludes.

2 Background

2.1 Robotic Tasks and Visual Models

vision information and fast visual model inference are important for the robotic tasks

Visual models often use local operators; introduce local operators and non-local operators.

2.2 Resource Limitations of Robots

Computation power; wireless network

2.3 Related Work

offloading methods; intra-DP

Cache related (Guan will take it)

3 System Overview

The chapter presents an overview of the design of CacheInf.

Working Environment and Metrics

We assume that the working scenario of CacheInf is a mobile robot performing robotic tasks in a real-world field which requires real-time visual model inference on the continuous image stream captured from the on-board camera, to achieve real-time response to various environment changes. The robot itself is equipped with low-power-consumption gpu to perform visual model inference which is slow and consumes too much power; it has wireless network access to a remote powerful GPU server that provides opportunity of acceleration, but the connection suffers from limited and unstable wireless network bandwidth.

While the requirements of real-time inference does not necessarily imply the requirement of high inference frequency, we measure the real-time metric by the average end-to-end inference latency when the robot is seamlessly performing inference, which leads to high inference frequency; the power consumption is also measured by average power

consumption to finish inference on each image in the same scenario.

3.1 Architecture of CacheInf

To reduce inference latency by reusing cached previous computation results to both reduce local computation time and transmission time of offloading, CacheInf basically consists of three blocks: a scheduler, a cache tracker and an executor (TODO fix the names).

Scheduler (TODO fix the names). During the initialization stage of the robotic task and CacheInf, CacheInf is granted access to the visual model and an initial input image and we mainly greedily pre-compute a schedule of various situations at this stage, since scheduling at runtime affects the real-time performance of the robotic task. We first profile the model at both the robot and the remote GPU server to gather information including shapes of the computation intermediates, the execution time of each operator (e.g., convolution, linear, etc.) on various scale of the input (e.g., from one tenth of the image to full scale of the image), the local property of each operator (i.e., whether the operator performs local computation) and so on.

Based on the above information, CacheInf finds sets of continuous local operators and assign the operators with smallest output sizes to be the operators to cache their computation results to reduce memory consumption of cache. Then we coarsely iterate through the possible wireless network bandwidth, distribution of cache between the robot and the server and the portion of reusable cache and greedily compute a plan of whether to compute on cache and the portion of local computation and offloaded computation at the server at the reduced transmission data volume reusing cache. We use the greedy strategy because we assume that both the wireless network bandwidth and the portion of reusable cache is unpredictable in the real-world scenario. Note that the precomputed schedule can be reused for a same visual model with the same settings.

3.2 Cache Tracker

At runtime, the selected operators at the previous stage will cache their computation results and the cache tracker identifies the reusable portion of such cached computation results. Given an input image, we extract and store its features using classic computation vision methods (we choose Flann algorithm in our implementation, which is state-of-the-art). For a current next image, we also extract and store its features and match them with the previous features (e.g., using KNN algorithm) and compute a perspective transform between the two images, which transforms the previous image such that the transformed previous image partially overlaps the current image and the non-overlapping areas are also marked. The features of the previous images is then discarded. The same transform can also be applied to the cached computation results since they are computed by local operators that keep the local geometries of the input image, and thus the reusable cached computation results are identified. Note that the computation involved in this process is light-weight compared with the visual model inference that typically involves hundreds of operators.

3.3 Executor (TODO fix the names)

The executor is responsible to actually select and execute an plan based on results of the above two processes at runtime. First, we further estimate the actual possible speedup by reusing cache, because the areas without cache needed for computation are often sparse and fragmented. We cluster the areas without cache into different nearest clusters and compute minimum bounding boxes for each of the clusters; then we greedily break up and recombine these bounding boxes to form a minimum new rectangle as a temporary input for the local operators and estimate its execution time based on its shape and the profile results from the initialization stage and select a precomputed plan for this input shape. If no evident speedup, we will ignore the cache and use the whole input.

With a selected plan where cache is enabled, the executor reuses the temporary input of reorganized areas without cache described above and feed it into the inference pipeline; it also handles the portion of local computation and the portion of offloaded computation to the remote GPU server. When appropriate, the executor breaks up the computation results of the temporary input and combines them with the transformed cache to recover geometries of the input image to get the correct result. With a selected plan where cache is disabled, the actions with cache involved are excluded, but note that in any cases, the cache at the remote GPU server is always reused to reduce transmission data volume.

4 Design

5 Implementation

6 Evaluation

6.1 Evaluation Settings

Testbed. The evaluation was conducted on a custom four-wheeled robot (Fig 2), and a custom air-ground robot(Fig 3). They are equipped with a Jetson Xavier NX [5] 8G onboard computer that is capable of AI model inference with local computation resources. The system runs Ubuntu 20.04 with ROS Noetic and a dual-band USB network card (MediaTek MT76x2U) for wireless connectivity. The Jetson Xavier NX interfaces with a Leishen N10P LiDAR, ORBBEC Astra depth camera, and an STM32F407VET6 controller via USB serial ports. Both LiDAR and depth cameras facilitate environmental perception, enabling autonomous navigation, obstacle avoidance, and SLAM mapping. The GPU server is a PC

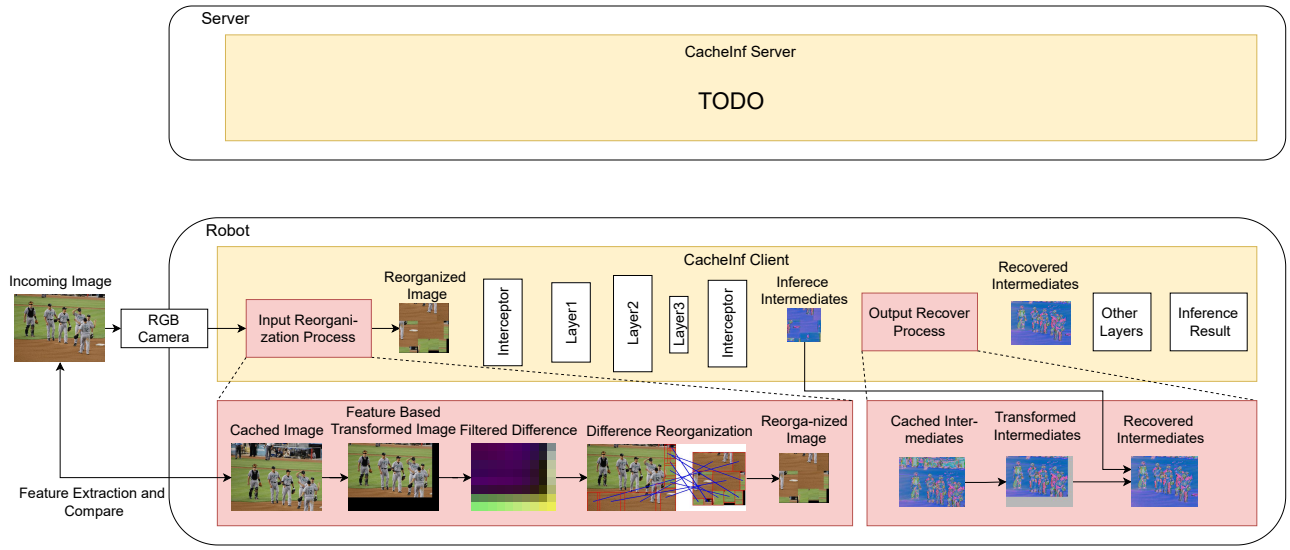


Figure 1. Architecture and working process of CacheInf. TODO...

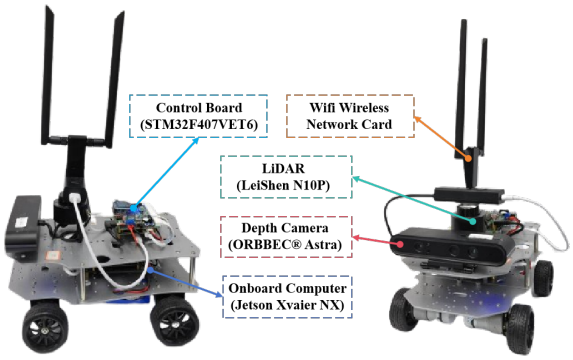


Figure 2. The composition of four-wheeled robot used in our evaluation.

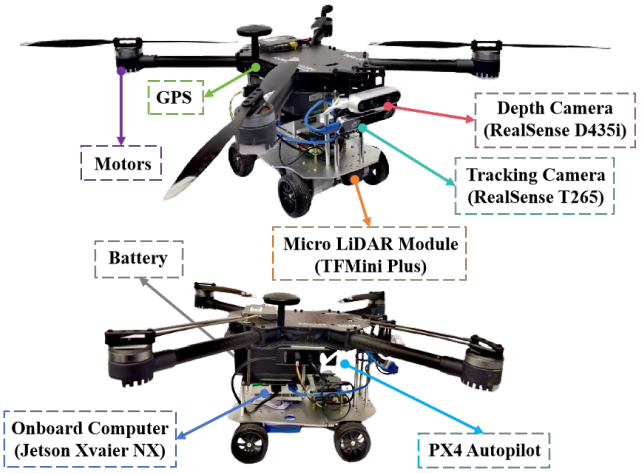


Figure 3. The composition of air-ground robot used in our evaluation.

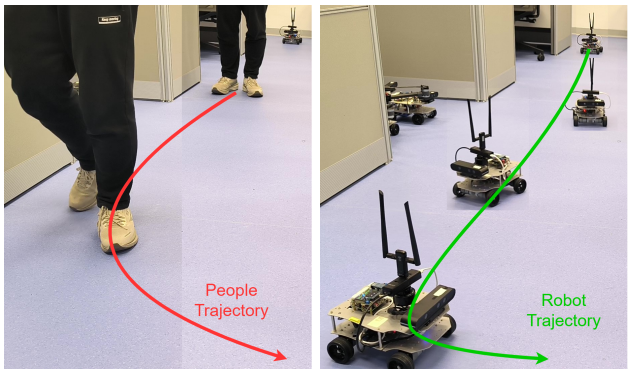
	inference	transmission	standby
Power (W)	13.35	4.25	4.04

Table 1. Power consumption (Watt) of our robot in different states.

equipped with an Intel(R) i5 12400f CPU @ 4.40GHz and an NVIDIA GeForce GTX 2080 Ti 11GB GPU, connected to our robot via Wi-Fi 6 over 80MHz channel at 5GHz frequency in our experiments.

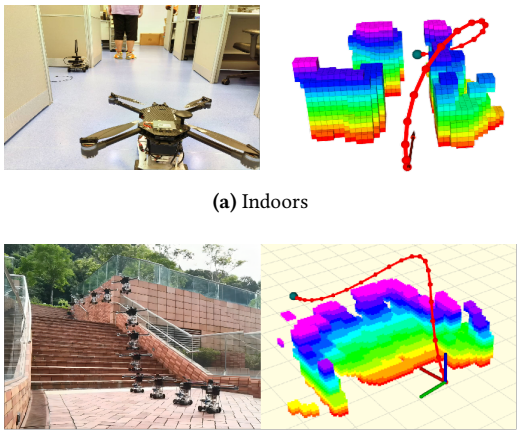
Tab. 1 presents the overall on-board energy consumption (excluding motor energy consumption for robot movement) of the robot in various states: inference (model inference with full GPU utilization, including CPU and GPU energy consumption), transmission (communication with the GPU server, including wireless network card energy consumption), and standby (robot has no tasks to execute). Notice that different models, due to varying numbers of parameters, exhibit distinct GPU utilization rates and power consumption during inference.

Workload. We evaluated two typical real-world robotic applications on our testbed: Kapao, a real-time people-tracking application on our four-wheeled robot (Fig 4), and AGRNav, an autonomous navigation application on our air-ground robot (Fig 5). These applications feature different model input and output size patterns: Kapao takes RGB images as input and outputs key points of small data volume. In contrast, AGRNav takes point clouds as input and outputs predicted point clouds and semantics of similar data volume as input, implying that AGRNav needs to transmit more data during



(a) Targeted people **(b)** Robot moving trajectory

Figure 4. A real-time people-tracking robotic application on our robot based on a state-of-the-art human pose estimation visual model, Kapao [4].



(a) Indoors

(b) Outdoors

Figure 5. By predicting occlusions in advance, AGRNav [9] gains an accurate perception of the environment and avoids collisions, resulting in efficient and energy-saving paths.

distributed inference. And we have verified several models common to mobile devices on a larger scale to further corroborate our observations and findings: DenseNet [2], VGGNet [7], ConvNeXt [10], RegNet [11].

Experiment Environments. We evaluated two real-world environments: indoors (robots move in our laboratory with desks and separators interfering with wireless signals) and outdoors (robots move in our campus garden with trees and bushes interfering with wireless signals, resulting in lower bandwidth). The corresponding bandwidths between the robot and the GPU server in indoors and outdoors scenarios are shown in Fig. **??**.

Baselines. We selected two SOTA inference acceleration methods as baselines: DSCCS [3], aimed at accelerating inference, and Hybrid-Parallel [8] (referred to as HP), that parallelizes local computation and offloading to further acceleratio inference. We also combined DSCCS with our cache mechanism (referred to as DSCCS-C) to present another perspective about CacheInf's performance gain.

The evaluation questions are as follows:

- RQ1: How much does CacheInf benefit real-world robotic applications by reducing inference time and energy consumption?
- RQ2: How does CacheInf perform on more models common to mobile devices?
- RQ3: How is the above gain achieved in CacheInf and what affects it?
- RQ4: What are the limitations and potentials of CacheInf?

6.2 End-to-End Performance on Real-World Applications

Inference Time.

Energy Consumption.

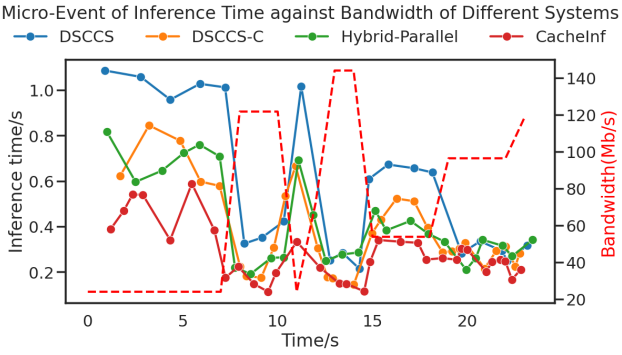


Figure 6. Micro-events about inference latency of different systems at different wireless network bandwidth.

6.3 Performance on Various Common Models

6.4 Micro-Event

6.5 Sensitivity

6.6 Sampling Rate of Video Frames

6.7 Discussion

7 Conclusion

References

[1] Wongun Choi, Khuram Shahid, and Silvio Savarese. What are they doing? : Collective activity classification using spatio-temporal relationship among people. In *Proc. of 9th International Workshop on Visual Surveillance (VSWS09) in conjuction with ICCV*, 2009.

[2] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

[3] Huanghuang Liang, Qianlong Sang, Chuang Hu, Dazhao Cheng, Xiaobo Zhou, Dan Wang, Wei Bao, and Yu Wang. Dnn surgery: Accelerating dnn inference on the edge through layer partitioning. *IEEE transactions on Cloud Computing*, 2023.

Model(number of parameters)	Local computation time/s	System	Transmission time/s		Inference time/s		Percentage(%)	
			indoors	outdoors	indoors	outdoors	indoors	outdoors
kapao(77M)	1.01(±0.03)	DSCCS	0.21(±0.1)	0.24(±0.12)	0.36(±0.2)	0.40(±0.17)	58.33	60.21
		DSCCS-C	0.18(±0.14)	0.22(±0.12)	0.33(±0.25)	0.37(±0.18)	66.67	67.57
		Hybrid-Parallel	0.24(±0.15)	0.28(±0.13)	0.31(±0.14)	0.34(±0.12)	77.42	82.35
		CacheInf	0.16(±0.13)	0.21(±0.18)	0.20(±0.16)	0.24(±0.20)	80.09	87.56
agrnav(0.84M)	0.60(±0.04)	DSCCS	0.10(±0.05)	0.15(±0.05)	0.41(±0.11)	0.47(±0.12)	24.39	31.91
		DSCCS-C	0.13(±0.07)	0.16(±0.06)	0.38(±0.10)	0.43(±0.13)	34.21	37.21
		Hybrid-Parallel	0.24(±0.08)	0.26(±0.07)	0.30(±0.09)	0.33(±0.07)	78.65	79.47
		CacheInf	0.18(±0.08)	0.20(±0.08)	0.21(±0.16)	0.25(±0.18)	86.71	80.01

Table 2. Average transmission time, inference time, percentage that transmission time accounts for of the total inference time and their standard deviation ($\pm n$) of Kapao and AGRNav in different environments with different systems. "Local computation" refers to inference the entire model locally on the robot.

Model(number of parameters)	System	Power consumption(W)		Energy consumption(J) per inference	
		indoors	outdoors	indoors	outdoors
kapao(77M)	Local	10.61(±0.49)	10.61(±0.49)	9.79(±0.03)	9.79(±0.03)
	DSCCS	6.38(±2.21)	6.63(±2.38)	2.30(±0.55)	2.65(±0.55)
	DSCCS-C	6.30(±2.15)	6.53(±2.12)	2.08(±0.50)	2.42(±0.53)
	HP	7.05(±1.63)	6.94(±0.98)	2.19(±0.62)	2.35(±0.42)
	CacheInf	7.53(±1.62)	7.30(±0.96)	1.51(±0.60)	2.75(±0.41)
agrnav(0.84M)	Local	8.11(±0.25)	8.11(±0.25)	4.86(±0.01)	4.86(±0.01)
	DSCCS	6.21(±1.50)	7.29(±1.55)	2.55(±0.19)	3.43(±0.18)
	DSCCS-C	6.17(±1.56)	7.00(±1.43)	2.34(±0.20)	3.01(±0.20)
	HP	7.52(±0.51)	8.04(±0.45)	2.26(±0.15)	2.63(±0.15)
	CacheInf	7.83(±0.57)	8.23(±0.56)	1.64(±0.17)	2.06(±0.16)

Table 3. The power consumption against time (Watt) and energy consumption per inference (Joule) with standard deviation ($\pm n$) of Kapao and AGRNav different environments with different systems. "Local" represents "Local computation".

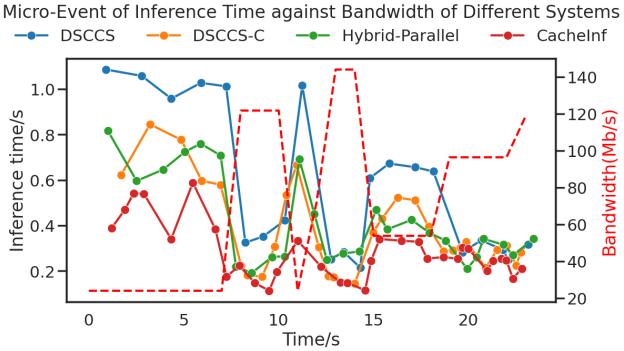


Figure 7. TODO Micro-events about inference latency of different systems at different cached ratio with fixed wireless network bandwidth.

[4] William McNally, Kanav Vats, Alexander Wong, and John McPhee. Rethinking keypoint representations: Modeling keypoints and poses as objects for multi-person human pose estimation. In *European Conference on Computer Vision*, pages 37–54. Springer, 2022.

[5] NVIDIA. The world's smallest ai supercomputer. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-series/, 2024.

[6] F. Perazzi, J. Pont-Tuset, B. McWilliams, L. Van Gool, M. Gross, and A. Sorkine-Hornung. A benchmark dataset and evaluation methodology for video object segmentation. In *Computer Vision and Pattern Recognition*, 2016.

[7] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[8] Zekai Sun, Xiuxian Guan, Junming Wang, Haoze Song, Yuhao Qing, Tianxiang Shen, Dong Huang, Fangming Liu, and Heming Cui. Hybrid-parallel: Achieving high performance and energy efficient distributed inference on robots, 2024.

[9] Junming Wang, Zekai Sun, Xiuxian Guan, Tianxiang Shen, Zongyuan Zhang, Tianyang Duan, Dong Huang, Shixiong Zhao, and Heming Cui. Agrnav: Efficient and energy-saving autonomous navigation for air-ground robots in occlusion-prone environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[10] Sanghyun Woo, Shoubhik Debnath, Ronghang Hu, Xinlei Chen, Zhuang Liu, In So Kweon, and Saining Xie. Convnext v2: Co-designing and scaling convnets with masked autoencoders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16133–16142, 2023.

Model(number of parameters)	Local computaion time/ms	System	Transmission time/ms		Inference time/ms		Percentage(%)	
			indoors	outdoors	indoors	outdoors	indoors	outdoors
DenseNet121(7M)	74.5(±18.7)	DSCCS	16.2(±40.9)	20.8(±51.9)	81.4(±27.2)	86.6(±27.7)	19.95	24.07
		DSCCS-C	20.4(±43.5)	25.8(±56.9)	85.5(±27.9)	89.6(±29.3)	23.86	28.80
		HP	53.4(±34.5)	52.9(±23.9)	74.5(±85.7)	55.1(±15.6)	71.70	96.05
		CacheInf	56.3(±37.5)	57.5(±43.5)	76.3(±90.6)	78.1(±33.6)	73.79	73.62
RegNet(54M)	175.0(±23.6)	DSCCS	47.6(±47.8)	60.5(±54.0)	77.8(±39.3)	86.2(±37.9)	61.22	70.22
		DSCCS-C	50.7(±49.8)	62.5(±53.6)	70.8(±33.3)	79.5(±39.2)	71.61	78.61
		HP	49.6(±21.7)	59.9(±23.4)	55.0(±24.8)	64.2(±25.2)	90.18	93.34
		CacheInf	44.2(±27.7)	48.5(±25.3)	45.3(±35.0)	49.2(±37.2)	97.57	98.58
ConvNeXt(88M)	160.2(±21.0)	DSCCS	46.9(±43.1)	56.7(±52.1)	72.4(±35.7)	84.7(±36.3)	64.78	66.95
		DSCCS-C	48.0(±45.0)	53.2(±50.1)	56.8(±28.1)	70.8(±39.0)	84.51	75.14
		HP	50.4(±32.2)	61.9(±34.8)	53.9(±26.2)	65.7(±27.7)	93.51	94.23
		CacheInf	40.7(±40.0)	50.7(±40.3)	46.7(±35.4)	56.8(±45.0)	87.15	89.26
VGG19(143M)	118.0(±18.9)	DSCCS	38.9(±47.1)	41.6(±53.8)	65.2(±28.1)	75.5(±27.1)	59.75	55.09
		DSCCS-C	42.7(±30.2)	52.0(±50.3)	53.2(±33.0)	60.3(±30.9)	80.26	86.24
		HP	44.8(±20.9)	51.5(±15.0)	47.6(±18.1)	53.6(±14.7)	94.15	96.07
		CacheInf	37.8(±31.2)	43.5(±13.2)	41.1(±20.3)	46.6(±12.8)	94.26	93.34
ConvNeXt(197M)	316.7(±31.0)	DSCCS	56.0(±36.1)	67.0(±37.6)	79.2(±35.9)	90.6(±35.4)	70.72	73.98
		DSCCS-C	56.0(±39.0)	63.0(±30.2)	64.7(±40.2)	68.6(±35.0)	86.55	91.84
		HP	56.4(±34.7)	66.5(±33.7)	59.7(±26.6)	68.0(±26.6)	94.43	97.88
		CacheInf	40.4(±37.8)	46.9(±40.0)	44.7(±33.3)	49.0(±30.8)	90.38	95.71

Table 4. Average transmission time, inference time, percentage that transmission time accounts for of the total inference time and their standard deviation ($\pm n$) of common AI models in different environments with different systems.

[11] Jing Xu, Yu Pan, Xinglin Pan, Steven Hoi, Zhang Yi, and Zenglin Xu. Regnet: self-regulated network for image classification. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

Model(number of parameters)	System	Power consumption(W) indoors	Power consumption(W) outdoors	Energy consumption(J) per inference indoors	Energy consumption(J) per inference outdoors
DenseNet121(7M)	Local	8.2(±0.27)	8.2(±0.27)	0.46(±0.04)	0.46(±0.04)
	DSCCS	6.91(±0.45)	6.86(±0.46)	0.56(±0.04)	0.59(±0.04)
	DSCCS-C	7.01(±0.43)	6.96(±0.43)	0.60(±0.07)	0.52(±0.06)
	HP	5.36(±0.79)	5.79(±0.24)	0.4(±0.06)	0.32(±0.01)
	CacheInf	6.01(±0.92)	6.31(±0.56)	0.46(±0.12)	0.49(±0.01)
RegNet(54M)	Local	9.0(±0.3)	9.0(±0.3)	1.37(±0.02)	1.37(±0.02)
	DSCCS	5.84(±1.79)	5.36(±1.34)	0.45(±0.14)	0.46(±0.12)
	DSCCS-C	6.04(±1.88)	5.96(±1.45)	0.43(±0.16)	0.47(±0.19)
	HP	5.24(±1.43)	5.28(±1.52)	0.29(±0.08)	0.34(±0.1)
	CacheInf	5.20(±1.51)	5.43(±1.77)	0.24(±0.08)	0.327(±0.09)
ConvNeXt(88M)	Local	9.7(±0.34)	9.7(±0.34)	1.34(±0.02)	1.34(±0.02)
	DSCCS	6.01(±0.27)	5.71(±1.56)	0.43(±0.05)	0.48(±0.13)
	DSCCS-C	6.20(±0.33)	5.91(±0.21)	0.35(±0.17)	0.42(±0.25)
	HP	6.68(±1.23)	6.68(±1.21)	0.36(±0.07)	0.44(±0.08)
	CacheInf	6.70(±0.55)	6.63(±0.26)	0.31(±0.07)	0.38(±0.08)
VGG19(143M)	Local	9.78(±0.34)	9.78(±0.34)	0.95(±0.02)	0.95(±0.02)
	DSCCS	6.58(±2.14)	6.93(±2.35)	0.43(±0.14)	0.52(±0.18)
	DSCCS-C	6.82(±2.10)	7.23(±2.45)	0.36(±0.18)	0.43(±0.30)
	HP	6.51(±1.74)	7.32(±1.52)	0.31(±0.08)	0.39(±0.08)
	CacheInf	6.70(±1.88)	7.22(±1.36)	0.27(±0.10)	0.34(±0.09)
ConvNeXt(197M)	Local	10.72(±0.38)	10.72(±0.38)	3.12(±0.03)	3.12(±0.03)
	DSCCS	5.06(±0.31)	5.02(±0.37)	0.4(±0.02)	0.45(±0.03)
	DSCCS-C	4.86(±0.44)	4.99(±0.39)	0.31(±0.05)	0.34(±0.09)
	HP	4.57(±0.23)	4.54(±0.25)	0.27(±0.01)	0.31(±0.02)
	CacheInf	5.26(±0.40)	5.39(±0.27)	0.24(±0.05)	0.26(±0.04)

Table 5. The power consumption against time (Watt) and energy consumption per inference (Joule) with standard deviation ($\pm n$) of common AI models in different environments with different systems. "Local" represents "Local computation".

Model	Statistics	Difference Filter Parameter (n)		
		50	70	130

Table 6. How different difference filter parameter (n) for identifying reusable cache affects the inference latency of CacheInf and the accuracy of visual models.

Model	Statistics	Sampling rate		
		1	2	4

Table 7. How the sampling rate of video frames influence the performance of CacheInf.