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

Abstract

Placeholder Placeholder Placeholder Placeholder

1 Introduction

Visual information is vital for various robotic tasks deployed on real-world edge devices (typically mobile robots), such as TODO. 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 suffer from limited and unstable wireless network bandwidth, making both local computation slow and naively offloading the computation to GPU servers slow.

To tackle this problem, we have the following two observations: 1. the robotic visual model mostly leverages local operators (i.e., computation results relies on local geometries); 2. the images for robotic visual model inference are typically continual. These imply that between visual model inference on consecutive images in visual robotic tasks, part of the previous computed results of local operators can be cached and reused, providing opportunity to both reduce local computation time and reduce transmission data volume, accelerating the overall visual model inference.

Based on these observations, 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 time.

The design of CacheInf is non-trivial. The first challenge is to transform cached results to local computation acceleration. While the cached result of overlapping areas can be reused and the corresponding computation can be skipped, the sparse and fragmented remaining area can not be computed on local operators which are typically highly optimized for dense local geometries and outperform their counterparts for sparse local geometries (e.g., TODO). To make use of these highly optimized local operators, in CacheInf we search for a minimal set of rectangles that covers the uncached areas, split and combine them to a new rectangle while preserving local geometries and use it as a temporary input for the local operators whose computed result can be broken up and combined with the cache to form the correct result. We also profile the cost of this process and would choose to ignore cache when the cached portion is too small to bring acceleration.

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 that the parameters of local operators and naively caching all the computation result can lead to XX times memory consumption in our evaluation.

We observe that in visual models, the computation result of a local operator often serves as the input of another local operator where local geometries are kept. Thus in the cached scenario, 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 of the previous computation result between the two operators can be abandoned to reduce memory consumption. We greedily search for a set of continual local operators whose starting and ending operators incur the least memory consumption when caching their computation results, so as to minimize cache memory consumption.

The third challenge is under various distribution scenarios of cache (e.g., all cache is located at local or the remote GPU server), how to fully utilize the cache for acceleration. For example, when all cache is located at local due to previous limited wireless network bandwidth and we currently have a suitable wireless network bandwidth to offload computation to the remote GPU server, directly offloading all of the input means abandoning all local cache, damaging the potential gain of cache.

To tackle this problem, we integrate a recent new offloading paradigm named Intra-DP (TODO: cite arxiv name): during visual model inference on an image, Intra-DP enables splitting of the input of local operators at the dimension of columns and assign 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. Under this paradigm, we can partially leverage the local cache to compute on a split of the input which leads to faster local computation and also reduced transmission data volume, to better utilize the existing cache. We extend the scheduler of Intra-DP to further consider the potential

acceleration with cache and combine the heuristics about the previous two challenges into a new scheduling algorithm named XXX.

We implemented CacheInf using python and pytorch on Ubuntu20.04. To efficiently combine uncached areas into a new rectangle and recover this rectangle to correct result, we implemented these two operations as cuda kernel using taichi for faster parallel computation on GPU. Our baselines include plain local computation, a state-of-the-art computation offloading system named TODO and our self-modified counterpart with cache enabled and Intra-DP. Our datasets include the standard video datasets of TODO from torchvision. 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 and a PC equipped with an Nvidia 2080ti GPU as the remote GPU server. 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 paradigm of using cached computation results to accelerate visual modified inference on a continuous stream of input images on robots and the corresponding scheduling algorithm TODO. The resulting system, CacheInf, accelerates both local computation on the robot and the data transmission while offloading the computation to the remote GPU server by collaboratively considering and reusing cached computation results on both the robot and the server. 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 TODO.

2 Background

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 Implementation

5 Evaluation

6 Conclusion

7 Conclusions

References