Low-latency Speculative Inference On Distributed Multi-modal Data Streams

Tianxing Li†, Jin Huang†, Erik Risinger†, and Deepak Ganesan†

†Department of Computer Science and Engineering, Michigan State University, East Lansing, MI
‡Department of Computer Science, UMass Amherst, Amherst, MA
litianx2@msu.edu, jinhuang@cs.umass.edu, erisinger@umass.edu, dganesan@cs.umass.edu

ABSTRACT
While multi-modal deep learning is useful in distributed sensing tasks like human tracking, activity recognition, and audio and video analysis, deploying state-of-the-art multi-modal models in a wirelessly networked sensor system poses unique challenges. The data sizes for different modalities can be highly asymmetric (e.g., video vs. audio), and these differences can lead to significant delays between streams in the presence of wireless dynamics. Therefore, a slow stream can significantly slow down a multi-modal inference system in the cloud, leading to either increased latency (when blocked by the slow stream) or degradation in inference accuracy (if inference proceeds without waiting). In this paper, we introduce speculative inference on multi-modal data streams to adapt to these asymmetries across modalities. Rather than blocking inference until all sensor streams have arrived and been temporally aligned, we impute any missing, corrupt, or partially-available sensor data, then generate a speculative inference using the learned models and imputed data. A rollback module looks at the class output of speculative inference and determines whether the class is sufficiently robust to incomplete data to accept the result; if not, we roll back the inference and update the model’s output. We implement the system in three multi-modal application scenarios using public datasets. The experimental results show that our system achieves 7 – 128x latency speedup with the same accuracy as six state-of-the-art methods.

CCS CONCEPTS
- Computer systems organization → Cloud computing.

KEYWORDS
Edge computing, cloud computing, computing off-loading, deep neural networks

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:

1 INTRODUCTION
The proliferation of low-power yet data-rich sensors such as cameras, microphones, LIDAR, hyperspectral imagers, and RF imagers, has led to increased use of these sensors on IoT devices. There is also growing evidence that multi-modal inference is more effective than using a single modality in applications such as speech recognition [61], 3D scene understanding [56], health monitoring [27], safety [3], material identification [23], augmented reality [59], robotics [46], and self-driving vehicles [13].

These advantages are also leading to distributed deployments of multi-modal systems, particularly for monitoring complex environments where monitoring needs to be done around corners, through walls, in low light, under foliage, or in other occluded settings [16, 78]. The ability to observe a scene from different vantage points allows us to reconstruct various physical properties of a dynamic scene independently from multiple sensing modalities. This allows us to extract a rich set of parameters about the physical scene from diverse sensors — for example, the location, displacement, speed, direction and material properties of objects may be inferred from various imaging modalities. These parameters can be leveraged to independently cross-check information from alternate sensors from different vantage points, which is particularly important in complex environments. Many applications exhibit such complexity including city-scale real-time analytics about the urban environment, infrastructure, and activity [43, 64], industrial IoT settings for predictive control and fault detection [63], and healthcare settings where data from body-worn sensors (e.g., Fitbit) need to be fused with environmental sensors (e.g., Wi-Fi access points) for measuring the contextual environment around the user [76].

An overarching challenge in these systems is how to process the distributed multi-modal data from these sensors. Advanced sensors are capable of producing extremely large and continuous data streams, and it is impractical to stream all collected data to the cloud for data fusion. At the same time, it is important to fuse these signals and leverage advances in neural networks that have significantly improved in their ability to combine data from multiple modalities.

While a number of recent efforts have been exploring the problem of partitioned machine learning models across IoT devices and the cloud to deal with this issue (e.g., [50]), we face additional challenges in the context of distributed multi-modal inference since data generation rates can vary widely across these data-rich sensor modalities. For example, a video stream requires one or two orders
We show that most classes are robust to such speculative inference waiting for all inputs to be completely available. Instead of blocking, work dynamics may achieve a better trade-off between accuracy and resource consumption (e.g., bandwidth) and accuracy. When the inference results are determined over lower-fidelity data, and only a few classes are more sensitive and then generate a speculative inference using the imputed data. In this paper, we introduce speculative inference on multi-modal data streams to adapt to constrained bandwidth and unpredictable network dynamics. Unlike prior approaches, our system does not modify the deep learning models used in existing multi-modal applications but enables speculative execution of these models without waiting for all inputs to be completely available. Instead of blocking the inference pipeline, our system leverages the fact that different modalities are complementary (e.g., they can provide additional information measured from the same object) and correlate temporally to impute missing, corrupt, or partially-available sensor data, and then generate a speculative inference using the imputed data. We present a novel architecture for multi-modal models that enables speculative inference to adapt to fluctuating network bandwidth and asymmetry in data sizes between modalities. We implement the system in three multi-modal application scenarios using public datasets to examine the feasibility and performance of speculative inference on multi-modal data streams. Experimental results show that the system achieves 7 ∼ 128× latency speedup with the same accuracy as six state-of-the-art methods, enabling fast and accurate multi-modal learning under fluctuating network bandwidth.

2 THE CASE FOR SPECULATIVE INFEERENCE

The notion of speculative inference is inspired by speculative execution in modern processors and OS kernels, which is highly effective.
We introduce speculative inference, which executes a multi-modal deep-learning pipeline without waiting for inputs from all modalities to be available. The system takes data from both fast and slow data streams as inputs, predicts features of the delayed data, speculatively executes the inference pipeline, and rolls back execution if the inference result is statistically likely to be wrong. Compared with state-of-the-art approaches [29, 35, 39, 84, 85], our method further reduces inference latency while maintaining good accuracy.

Figure 1 shows the main modules involved in speculative inference over a multi-stream, multi-modal data pipeline, including a data imputation module, a rollback module, and a data alignment module. The data imputation module leverages a conditional Generative Adversarial Network (GAN) to reconstruct features of any missing, corrupt, or partially-available sensor data. The rollback module provides a lightweight rollback mechanism that leverages the fact that many classification categories do not need data from all modalities to be present at high fidelity, and only a small number of categories are sensitive to the effects of speculative inference. The module, therefore, learns how robust the different classification categories are to the availability of data from different modalities, and triggers the rollback operation judiciously. The data alignment module aligns multi-modal data streams without using timestamps or injecting any synchronization pulses or frames, which is more general than state-of-the-art methods [1, 41, 86]. This module leverages a series of generative models to generate (a) intermediate features using the faster data stream (e.g., audio) only and (b) a series of intermediate features using both fast and slow data streams with different delay configurations. It aligns the two data streams by searching for the minimum $L1$ distance between (a) and (b).

**Applications:** Speculative inference can benefit data-driven applications that involve the fusion of multi-modal data under latency constraints. Wirelessly networked distributed smart camera and microphone systems are common in security, automation, biometric and environmental applications [22, 81]. Speculative inference is an effective way to fuse such multi-modal signals in real-time while being robust to bandwidth differences. Wireless health is another domain where speculative inference can be useful. For example, an adverse health event detection system might fuse information from a body-worn sensor like a smartwatch and a remote monitoring sensor at home to determine the likelihood that a serious event has occurred [76]. Real-time actuation can also benefit from these techniques. For example, robotic navigation or grasping in occluded conditions can leverage multiple modalities from different vantage points to complete difficult tasks in complex environments [24].
and the data alignment module. Together, these modules optimize the accuracy of multi-modal models while minimizing end-to-end latency under variable network dynamics.

We focus on the two-modality case (e.g., video and audio) because there is a dearth of distributed multi-modal datasets involving more than two modalities [8]. We expect that our methods will generalize to more modalities, but we leave this to future work when labeled datasets of such scenarios are more widely available. In this work, we assume that data from the two modalities is transmitted from two separate devices to the cloud using any common network protocol (e.g., TCP, UDP). We also assume that the IoT clocks are not always going to be time synchronized, either because time synchronization has been turned off to reduce resource usage or because the clock crystals used on these devices have a high drift-rate, resulting in transient loss of synchronization.

3.1 Data Imputation Module

We propose the use of a conditional GAN to predict the features of missing or delayed input data from one modality. A GAN trains two separate neural networks simultaneously: a generator \( G \) learns to map from a latent space to a data distribution of interest, and a discriminator learns to distinguish between real samples and synthetic samples produced by the generator. To effectively train the generator, the discriminator provides feedback to the generator about the quality of the synthetic samples, and the generator imputes better samples to try to fool the discriminator. During training, the generator and the discriminator play a competitive game, resulting in synthetic samples that are very close to real samples. A vanilla generator usually takes a noise vector as input, but when conditioned generation is needed, the noise vector can be an explicit source [60]. The generator then leverages receptive fields to denoise and densify the input samples by referring to the generator’s spatial or temporal contexts [51]. In our system, the generator leverages data from both fast and slow streams to learn and predict the features of one modality from the raw data of another.

Directly applying the GAN in our system does not work well due to the difference in data sizes and underlying sensing modalities. For example, a raw audio stream is a 2D vector and contains rich temporal information, whereas raw video is a 3D vector which contains rich spatial and temporal information. Also, the noise distributions of audio and video data are different due to inherent differences between the sensors and their respective noise sources. Therefore, training a GAN that is capable of generating realistic raw sensor data from partially received multi-modal sensor data is extremely challenging. Recent efforts have made some progress in generating images from audio signals [14, 74]. But the predicted images lose significant detail compared with the originals, which degrades performance in predictive applications.

To address this challenge, we propose to impute missing or incomplete data by generating intermediate feature maps, rather than attempting to generate realistic raw sensor data. This method is inspired by computer vision applications in which intermediate features from pre-trained deep neural networks are used to guide the generator in a GAN [9, 37, 75]. Figure 2 illustrates a typical audio-visual multi-modal model that has been widely used in the computer vision and deep learning communities [32, 68], which takes raw sensor data from a camera and a microphone as inputs, and computes intermediate feature maps \( F_{\text{camera}} \) and \( F_{\text{audio}} \) via the feature extraction layers \( f_{\text{camera}} \) and \( f_{\text{audio}} \) respectively. A fusion layer, \( g \), concatenates \( F_{\text{camera}} \) and \( F_{\text{audio}} \) to generate an intermediate feature map \( F_{\text{concat}} \). Finally, the prediction layer \( h \) uses \( F_{\text{concat}} \) to compute the final output, which can be derived by Equation 1.

\[
\text{Output} = h(g(f_{\text{camera}}(I_{\text{camera}}), f_{\text{audio}}(I_{\text{audio}))))
\]  

Since the bandwidth requirement for audio is less than that of video, it is typically the faster of the two data streams to arrive for inference in the cloud. Thus, our system aims to impute the features of the slower video stream. We design our GAN models to leverage the features from audio and incomplete video data to generate synthetic feature maps of the full video stream. Specifically, we use audio features \( F_{\text{audio}} \) and features \( F_{\text{camera}} \) from video data that is incomplete due to transmission delays to mimic the features \( F_{\text{camera}} \) of the complete video stream. Figure 3 shows an overview of our data imputation module.

We use the intermediate feature map \( F_{\text{camera}} \) prior to the fusion layer as the input to the discriminator because \( f_{\text{camera}} \) acts as a feature extraction and denoising module, so \( F_{\text{camera}} \) has a higher signal-to-noise ratio than the raw sensing data \( I_{\text{camera}} \). We leverage the feature map \( F_{\text{concat}} \) after the fusion layer \( g \) as the generator’s
input to better utilize the correlated cross-modal information, because simply concatenating the intermediate feature maps $F_{\text{camera}}$ and $F_{\text{audio}}$ does not extract correlations between audio and visual content. We observe that many multi-modal models use Long Short Term Memory (LSTM) networks or Bidirectional Gated Recurrent Unit (BGRU) networks in the fusion layer, which jointly models temporal dynamics [32, 68].

**Training of GAN models** The objective function considers losses from two sources: a conditional GAN, $L_{\text{GAN}}$, and a perceptual loss, $L_{p}$. We leverage a conditional GAN to model the conditional distribution of the real intermediate feature map $F_{\text{camera}}$ from the complete video input $I_{\text{camera}}$ given the intermediate feature map $F_{\text{concat}}$, which is generated from the full audio data $I_{\text{audio}}$ and partial video data $I_{\text{camera}}$. The conditional GAN loss can be expressed as:

$$F_{\text{camera}} = f_{\text{camera}}(I_{\text{camera}})$$  

(2)

$$F_{\text{concat}} = g(f_{\text{camera}}(I_{\text{camera}}); f_{\text{audio}}(I_{\text{audio}}))$$  

(3)

$$L_{\text{GAN}}(G, D) = \mathbb{E}_{\text{camera}}[\log D(F_{\text{camera}, x})] + \mathbb{E}_{\text{concat}}[\log(1 - D(G(F_{\text{concat}, x})))$$  

(4)

where $y$ is the ground truth and the generator $G$ tries to minimize this objective function against an adversarial discriminator, $D$, that tries to maximize it [38]. Then, we follow [40, 51] and introduce a perceptual loss, $L_{p}$, in our objective function to compare high-level differences and stabilize GAN training:

$$L_{p}(G) = \mathbb{E}_{n=1}^{N} ||V^{(n)}(G(F_{\text{concat}})) - V^{(n)}(F_{\text{camera}})||_{1}$$  

(5)

where $V$ is a pre-trained VGG network that helps to quantify the perceptual differences of the content between our intermediate feature maps. In the VGG network, each layer $n$ measures different levels of perception.

Our final objective, therefore, is

$$G^{*} = \arg\min_{G} \max_{D} L_{\text{GAN}}(G, D) + \lambda L_{p}(G)$$  

(6)

where $\lambda$ is a hyper-parameter for regularization.

**Offline Training** We first extract intermediate feature map $F_{\text{camera}}$ within the multi-modal model. $F_{\text{camera}}$ will then be the discriminator’s input. We next compute the concatenated feature map $F_{\text{concat}}$ within the learning model as the generator’s input. We modify the quality (e.g., resolution) and the quantity (e.g., delay time) of video data $I_{\text{camera}}$ and train a series of GANs ($G_{ij,k}$, $i \in [1, n], j \in [1, r], k \in [1, d]$) to adapt to fluctuating network bandwidths, where $n$, $r$, $d$ are the number of frame rate settings, resolution settings, and delay configurations, respectively. The definition of the delay configuration $k \in [1, d]$ is the number of the video frames received when audio data is fully received. Therefore, the number of GANs is $n \times r \times d$. For example, for the speech-recognition model in Sec. 5.1, the number of GANs is $4 \times 2 \times 25 = 200$. The data sizes of the intermediate features that our models learn to reconstruct are significantly smaller than those of the raw data, resulting in significantly lower training overhead than the state of the art [28, 66]. Experimental results in Sec. 5.6 show that our system reduces the training overhead by 80–90%, while still retaining good accuracy. For each resolution setting $j \in [1, r]$, we use an Image Expansion to transform the input resolution to a model-specific resolution (e.g. $256 \times 256$ for speech recognition, $224 \times 224$ for event detection); for each delay configuration $k \in [1, d]$, we pad the input with empty frames to reach the full frame count per sample (e.g. 29 frames for speech recognition, 300 frames for event detection).

Our system does not require the retraining of models pre-trained for individual inference tasks. Rather, it utilizes pre-trained models by replacing $F_{\text{camera}}$ with $F_{\text{camera}}^{*}$, which is the output of the generator $G$. As a result, our system can also be used to impute other sensing modalities (e.g., accelerometer) by retraining the set of GANs, which only takes 8–20 hours while retaining high accuracy.

**Online Inference** As sensing data arrives in the cloud, our system first uses the data alignment module in Sec 3.3 to align the data streams. Then it uses the corresponding GAN $G_{ij,k}^{*}$ to compute a synthetic feature map $F_{\text{camera}}$ via Eq. 7. Since only a single GAN needs to be loaded into memory at one time, the resource overhead is very small (< 100 MB memory in the GPU). The speculative inference can be derived via Eq. 9.

$$F_{\text{camera}} = G_{ij,k}^{*}(g(f_{\text{camera}}(I_{\text{camera}}^{'}); f_{\text{audio}}(I_{\text{audio}}^{'})))$$  

(7)

$$F_{\text{audio}} = f_{\text{audio}}(I_{\text{audio}})$$  

(8)

$$Output = h(g(F_{\text{camera}}^{*}, F_{\text{audio}}))$$  

(9)

### 3.2 Rollback Module

So far, we have described how the system imputes incomplete data in the slow data stream and speculatively infers the class output without waiting for all inputs to be available. But what do we do if the speculative inference is insufficiently accurate to use in a particular application? The rollback module helps our system to achieve a critical balance between high accuracy and latencies incurred by discarding already-computed inferences. The key question is: how can the speculative inference module ensure that its output is suitable for downstream processing, and when should it decide to roll back and wait for more (or higher-quality) data? To ensure that latency is minimized, this decision must be made rapidly, without waiting for all inputs to arrive.

A simple observation inspires our design: some prediction classes are resilient to incomplete data in one sensor modality, whereas others are more sensitive to incomplete or lower-quality data. Resilience to data quality issues depends on both the duration and the complexity of a classifiable event. For example, based on the complexity of lip movement, an audiovisual lipreading model can recognize the word “under” with less visual data than is required for the word “allegations.” Due to event duration, an audiovisual event detector can recognize “dog bark” with less visual data than that for “toilet flush.” For activity recognition, some activities like sitting and standing are easier to infer with incomplete data than others, such as discriminating between modes of transportation. Therefore, the rollback module will green-light predictions for certain inference classes if the speculative inference is statistically likely to achieve the desired accuracy. For other inference classes, the rollback module will hold onto the inference and wait for additional data to verify the output.

To validate this observation, we conduct an experiment using two public datasets and multi-modal models in Sec. 5. In this experiment, we enforce various delay configurations on the validation data.
and measure the minimum data requirements to achieve the best accuracy. Figure 4 shows the CDF of the minimum ratio of full video data to achieve the highest accuracy. We make two observations:

- For about 50% of the inference classes, the speculative inference can achieve the best accuracy with less than 50% of the full video data in both application scenarios. This validates our assumption that most classes are robust to the speculative inference over low fidelity data. For these inference classes, if the cloud has already received half of the total video frames in a prediction window (e.g., one second for speech recognition and ten seconds for event detection), the rollback module can release the lock and output the result of the speculative inference.

- For less than 10% of inference classes, the cloud needs to receive over 90% of total video frames to achieve the best accuracy in both application scenarios. For these inference classes, if the cloud receives less than 90% of the total video frames, it will discard the speculative inference and wait for the additional video frames. Once the cloud receives the additional data, the rollback module will re-compute and output the inference.

Our system considers latency as the main optimization objective and accuracy as the constraint. The objective of the rollback module is shown in Eq. 10:

\[
\mathbf{I}_{\text{camera}}^l, s^* = \arg \min \left\{ \text{Size}(\mathbf{I}_{\text{camera}}) \right\}
\]

\[
s.t. \quad \text{Acc}_{\max} = \text{Acc}(G_{ij}, \mathbf{I}_{\text{audio}}, \mathbf{I}_{\text{camera}} + \mathbf{I}_{\text{camera}}^l) \leq x\%
\]  

(10)

where \(G_{ij}\) is the GAN model, \(\mathbf{I}_{\text{audio}}\) is the audio data, and \(\mathbf{I}_{\text{camera}}\) is the delayed video data; \(\mathbf{I}_{\text{camera}}^l\) is additional video data acquired by the rollback module, and the total length and specifications are \(l\) and \(s\). The rollback module aims to achieve the desired accuracy (\(x\%) less than the maximum accuracy \(\text{Acc}_{\max}\) with minimum data size \(\text{Size}(\mathbf{I}_{\text{camera}})\).

Alternatively, we can also optimize accuracy against a desired latency. The objective function is shown in Eq. 11:

\[
\mathbf{I}_{\text{camera}}^l, s^* = \arg \min (1 - \text{Acc}(G_{ij}, \mathbf{I}_{\text{audio}}, \mathbf{I}_{\text{camera}} + \mathbf{I}_{\text{camera}}^l))
\]

\[
s.t. \quad \text{Size}(\mathbf{I}_{\text{camera}}) \leq y
\]

(11)

where \(y\) is the maximum number of additional frames after the speculative inference.

Algorithm 1 shows the details of the rollback module using the objective function in Eq. 10. Once the cloud receives the audio data \(\mathbf{I}_{\text{audio}}\) and partial video data \(\mathbf{I}_{\text{camera}}\), it counts the available video frames \(m\) in the current time window. The module then uses the audio data \(\mathbf{I}_{\text{audio}}\), \(m\) image frames, and \(m\) generators \(G_{ij}\), each of which uses the intermediate result \(\mathbf{I}_{\text{camera}} + \mathbf{I}_{\text{camera}}^l\) generated from the audio data and the video data \(\mathbf{I}_{\text{camera}}^l\) from \(k\) frames. We compute \(m\) inference classes and generate the speculative inference via voting. For this we utilize an accuracy table \(\text{AccTable}\), evaluated on validation data, to check if the speculative inference achieves the desired accuracy. The inference result will be accepted if the accuracy of the speculations on which that output depends has historically been above the desired accuracy or if the bandwidth is sufficient. Otherwise, based on the current network bandwidth, the cloud uses \(\text{AccTable}\) to search for the next spatial configuration and frame count that will achieve the desired accuracy while minimizing the size of transferred data. Finally, the cloud rolls back the inference, waits for additional data, and updates the model’s inference result when sufficient data is available.

The rollback module assumes that the distribution of the testing data is similar to the distribution of the validation data. However, we observe that if the test and validation data are limited (e.g., less than 5% of data is available to train \(\text{AccTable}\)), the rollback module tends to be erroneous. We also observe that the distribution of input data may vary over time. For these scenarios, the rollback
module can periodically update AccTable in the cloud when as data becomes available.

3.3 Data Alignment Module

This module aims to align multi-modal data streams without using timestamps or injecting synchronization pulses, as the former can be incorrect due to sensor failure or clock drift [83] and the latter may not be available under certain scenarios (e.g., low light or noisy environments). The key idea is to re-use the data imputation module in Sec. 3.1 to generate (a) a synthetic feature map $F^\ast_{camera,0}$ generated by audio-only data, and (b) a series of synthetic feature maps $F^\ast_{camera,k}$ generated by both audio and video data with multiple delay configurations $k$. $F^\ast_{camera,0}$ and $F^\ast_{camera,k}$ can be derived by Eq. 12 and 13.

$$F^\ast_{camera,0} = G(g(f^{audio}(I^{audio}))) \quad (12)$$

$$F^\ast_{camera,k} = G_k(g(f^{camera}(I^{camera,k}), f^{audio}(I^{audio}))) \quad (13)$$

We observe that when the delay configuration $k \in [1,d]$ matches the corresponding generator $G_k$, the output is close to $F^\ast_{camera,0}$. Specifically, when the delay configuration $k$ matches the corresponding generator $G_k$, the corresponding synthetic feature map $F^\ast_{camera,k}$ should be close to the real feature map $F_{camera}$. In this case, the distance between $F^\ast_{camera,0}$ and $F^\ast_{camera,k}$ is at a minimum. Therefore, we search for the minimum $L1$ loss of the two feature maps $F^\ast_{camera,0}$ and $F^\ast_{camera,k}$ among all possible delay configurations $k \in [1,d]$ to find the best alignment $k^\ast$ and align multi-modal data streams, which can be derived in Eq. 14. We use this observation to design a new method for aligning multi-modal streams without using timestamps.

$$k^\ast = \arg \min_{k \in [1,d]} ||F^\ast_{camera,0} - F^\ast_{camera,k}||_1 \quad (14)$$

Figure 5 shows some examples of data alignment in our system. For all examples, the $L1$ loss is minimized when the delay configuration matches the corresponding generator. We also observe that all the curves are concave downward. When the two data streams are not well aligned, the intermediate feature map tends to be erroneous, which increases the $L1$ distance. And the error increases with more misalignment between the two streams, leading to a larger $L1$ distance. To further improve the efficiency of data alignment, we apply a binary search algorithm to find the minimum $L1$ loss. Each alignment is an independent operation, so the alignment error will not accumulate. Our module assumes that the delay between two streams is less than a full window, e.g., the system needs to receive at least one frame from both streams within the same inference window. If the delay is larger than a full window, the system will wait until it receives one frame from the slow stream or discards the alignment task after the timeout.

One-time computational cost Executing a GAN for each possible delay configuration has a high cost, but we note that this cost is only incurred the first time that streams need to be fused. Once the synchronization offset has been determined, only minor adjustments are needed to mitigate the effects of clock drift over time. For example, a 15ppm clock crystal will drift by roughly 50 ms/hr, so infrequent re-synchronization can be done by looking for a few delay configurations around the current minima to obtain a new delay configuration.

4 IMPLEMENTATION

Overview We implement our system on a server utilizing two GTX-1080Ti GPUs, 32 GB RAM, and a Xeon E5-2620 v3 2.40 GHz CPU, which serves as a stand-in for the cloud. We use two Raspberry Pi 4Bs as IoT devices, one to send fast-stream data (e.g., audio) and the other to send slow-stream data (e.g., video). Execution of the GAN and inference models takes place on the cloud; the IoT devices are only used for data pre-processing and transmission. Thus, during the experiment we profile and benchmark the latency for GAN models and inference models on the cloud side.

IoT devices In our setup, both of the IoT devices connect to the cloud via either Wi-Fi or Bluetooth Low Energy (BLE) for data and control-signal transmission. We use Linux command wondershaper to directly limit upload bandwidths based on real-world traces of network dynamics to run evaluations under different bandwidth settings. We emulate the video streaming process by buffering raw camera sensor data on the IoT devices and periodically extracting and transmitting a new camera frame.

Video Encoding and Decoding We use the standard libav library from ffmpeg for video encoding on IoT devices and decoding in the cloud. The rollback module requires additional, lower-resolution frames and a specific range of frames. The specification of the resolutions and number of frames is sent from the cloud and received by the IoT devices. After receiving the specifications, the IoT devices re-initialize the encoder code AVCodecContext and discard all the encoded frame packets in the buffer. We maintain another buffer of raw input data so that the discarded encoded frame can be re-encoded at a lower resolution.

Test-bed Execution We duty-cycle the IoT nodes to better emulate a distributed sensor scenario to which speculative inference and rollback are well-suited. During cycle-off periods, the audio node sleeps, awaiting the execution of the rollback module if necessary. During cycle-on periods, both the audio and video nodes stream continuously to the cloud. The duration of the cycle-on period varies across tasks: the cycle-on period for speech recognition is 1 second, but for event detection, the active period is 10 seconds. Due to asymmetric data rate requirements, audio data is fully transmitted while only the partial video data is transmitted at the end of each active period. On the cloud side, we align the partial video data and full audio data to generate inference results, which are compared against a statistical accuracy threshold to determine whether to perform a rollback. Following the determination of the rollback module, the cloud sends the specifications and number of additional video frames back to the video node, then waits until the cloud receives the additional frames. Finally, we compute the final output based on both the initial and additional data.

5 EVALUATION

We start by describing the datasets we use and baselines we compare against, followed by a comprehensive evaluation.
5.1 Datasets and setup

We look at three application scenarios — speech recognition, event detection, and activity recognition — that require high accuracy with minimal inference latency. We run our system on three multimodal models using public datasets [17, 67, 68]. As described in Sec. 2.2, we assume that sensor data for each modality is streamed continuously from a separate device to the cloud during active, cycle-on periods. We then describe the four baselines — naive blocking, adaptive bitrate streaming, one-time profiling, and periodic profiling — that we compare against.

Speech Recognition Dataset We use a large Lip Reading in the Wild (LRW) dataset [17] consisting of 500 English word classes [5, 18]. Many words are either visually or verbally similar, making the words harder to recognize by a single modality only. The dataset consists of over 1,000 individual short video clips (1.16 seconds) from BBC news and talk shows for each word class; hundreds of different speakers are present in the videos. The LRW clips have been cropped to extract speakers’ faces only. The native resolution of the original videos is 1080p or higher, with the resolution of the LRW video cropped down to 256×256. The video frame rate is 25 FPS. The audio sampling rate is 44.1 kHz.

Event Detection Dataset We use the Audio-Visual Event (AVE) dataset, containing over 4,000 10-second videos covering a wide range of 28 audio-visual events (e.g., church bell ringing, dog barking) and domains (e.g., human activities, musical performances) [68]. There are at least 60 videos for each event category, and over half of the videos contain audio-visual events that span the full ten seconds. Each video contains ten labels (one label per second). The native resolution of the original video varies from 360p to 720p, and the authors resized the image resolution to 224×224 to extract visual features. The video frame rate is 30 FPS. The audio sampling rate is 44.1 kHz.

Heterogeneity Activity Recognition Dataset We use the STISEN dataset [67] comprising 43 million accelerometer and gyroscope readings from nine users performing six activities: biking, sitting, standing, walking, stair up, and stair down. Readings were recorded from four smartwatches, 31 smartphones and one tablet, representing 13 device models from four manufacturers. Note that, unlike audio-visual data, inertial data is not considered high-bandwidth; however, BLE radios are highly duty-cycled to minimize energy consumption, so delays of tens to hundreds of millisecond still occur.

Multi-modal Models We leverage three end-to-end multi-modal models for speech recognition [61], event detection [68], and activity recognition [62]. The speech recognition model leverages two separate residual networks (ResNet [32]) to extract features from raw images and audio signals, and two two-layer BGRUs to model the temporal dynamics of each modality. Two cascaded BGRU layers fuse the video and audio data by concatenating the features of each modality. Finally, a softmax activation function is used to predict the inference class. The event detection model utilizes two pre-trained CNNs to extract visual and audio features. Two LSTMs model temporal dependencies between the two modalities. A multi-modal fusion network, based on Multi-modal Residual Network [44] is used to generate a joint audiovisual representation. Finally, a fully-connected layer with a softmax activation function is used to classify audiovisual events. For activity recognition, each sensing modality has a dedicated CNN to extract preliminary features. These features are combined through fully-connected layers to infer the output.

Evaluation Metrics Performance is evaluated by two metrics: accuracy and latency. The best accuracy \( \text{Acc}_{best} \) is defined as the accuracy achieved using all data from both modalities within a given time window. End-to-end latency is computed by subtracting the timestamp of the final output from the timestamp recorded when incoming data from the faster stream (e.g., audio) reached the cloud for processing.

Baselines We compare our system with six baselines. The first baseline is naive blocking, in which clients transmit the highest data quality possible (full resolution and maximum frame-rate), and the inference pipeline simply blocks until all inputs from all data streams have arrived. The second baseline is adaptive bitrate streaming (ABS), widely used in live video streaming [10, 54]. This experiment requires the client to measure bandwidth in real time and vary the streaming video resolution to minimize latency.

We also compare our system against three existing video analytics methods. One is based on frame sampling, and the other two are profiling methods [39, 84]. Frame sampling minimizes the bandwidth required to transmit data by filtering out redundant information (e.g., images with fewer objects) [35, 85]. For a fair comparison, we choose a key-frame sampler that streams data without buffering or batch processing. Specifically, we sample key frames within an inference window using optical flow [79]. We then apply linear imputation to fit within dynamic bandwidth constraints. Profiling optimizes video streaming configurations to minimize the inference latency while achieving a desired accuracy. In general, state-of-the-art profiling methods can be further classified into two categories: one-time profiling [29, 84] in which the processing pipeline configuration is profiled once when video streaming begins, after which configuration is fixed, and periodic profiling [39, 80] in which the configuration is profiled periodically to find an optimal
accuracy-latency trade-off at the cost of profiling overhead. In our experiment, we do not take the profiling overhead into account.

Finally, we compare our system with an approach that retrains the multi-modal models themselves with varying amounts of stream lag. The number of the retrained models is equal to the number of delay configurations. During the test phase, the approach will measure the delay and select a retrained model that leverages the partial data to get the outputs without blocking the inference pipeline.

Configurations Vision algorithms often contain various configurable parameters or “knobs.” We focus on two knobs: frame rate and image resolution. For the LRW dataset, the frame rates are 25 FPS or 15 FPS, and the resolutions are 256×256, 128×128, 64×64, or 32×32. For the AVE dataset, the frame rates are 30 FPS or 15 FPS, and the resolutions are 224×224, 112×112, 56×56, or 28×28.

5.2 End-to-end Improvement

We start with the end-to-end improvement of our system over the six baselines. In this experiment, we do not consider the impact of encoding and compression at the IoT devices, which will be shown in Sec. 5.7. For our method, we leverage the data alignment module in Sec. 3.3 to align the data streams. For the baselines, we assume the data streams are perfectly aligned, as other techniques cannot align data streams without using timestamps or injecting synchronization pulses.

Synthetic trace-driven evaluation To demonstrate our system’s ability to adapt to asymmetric bandwidths across different modalities, we artificially vary bandwidths to simulate fluctuating network conditions using synthetic network traces generated with Pensieve [55], which uses a Markovian model in which each state represents an average throughput in the range from 200–800 Kbps. Each bandwidth value was drawn from a Gaussian distribution centered around the mean bandwidth with variance uniformly distributed between 0.05–0.5. The granularity of the network traces is one second.

Real-world trace-driven evaluation We now evaluate our system against several real-world bandwidth traces. Specifically, we look at multi-path TCP traces that were collected in a complex city environment [20]. The dataset covers the traffic produced by more than ten users using Nexus 5 smartphones. The Nexus 5
smartphones ran a modified Linux kernel (includes Multi-path TCP v0.89.5) to collect both Wi-Fi and cellular traces on Android 4.4. Since we assume that data from each modality is continuously transmitted from a separate device to the cloud, we randomly select two traces for the microphone node and the camera node. The network bandwidth fluctuates from 100 Kbps to 1 Mbps with a state-transition granularity of one second.

Figure 7 shows the comparison between our system and the four baselines on real-world network traces, and Table 1 shows the latency breakdown across different modules. The results are similar to those attained with synthetic traces. ABS, frame sampling, retrained model achieve the minimum end-to-end latency (roughly 60ms lower than our method) but at significant accuracy loss (up to 12% accuracy loss). Also, the training overhead of the retrained model method (100–250 hours) is about ten times higher than our system (8–20 hours), as our system only imputes intermediate features rather than the raw sensor data and it does not modify the existing inference model. We show accuracy comparable to one-time and periodic profiling but offer 7.5–10x, and 7–8x latency speedup. We have 1% lower accuracy than naïve blocking but 24–120x latency speedup. Since our system does not optimize data transmission configurations before the latency is detected on the cloud, it is also possible to integrate profiling and frame sampling methods into our system to further improve inference efficiency.

Generalizability Across Modalities While most of our results show performance on audiovisual (LRW [17] and AVE [68]) datasets, we briefly demonstrate that our technique generalizes to other sensor modalities. In this experiment, we evaluate our system on a multi-modal model for activity recognition with accelerometer and gyroscope data [62].

Figure 8 compares our approach against naïve blocking. (Results against other baselines are similar to the above observations.) We introduce various delays (0–1.25s) to one of the sensor streams corresponding to duty-cycled radio transfer delays. Similar to previous results, our system achieves 7–24x latency speedup over naïve blocking, with accuracy drops of only 1.4%, even when a 1.25s delay to one modality is introduced. Therefore, our system is quite robust and can be generalized to various sensing modalities.

5.3 Impact of Data Imputation

Next, we microbenchmark the impact of individual components in our system. We start with our data imputation module and investigate the impact of different GAN networks. Our system applies an image-to-image translation framework Pix2Pix [38] to impute intermediate features. We also implement our system on two commonly used generative networks: BicycleGAN [89] and Pix2PixHD [77]. BicycleGAN is a popular image-to-image multimodal translation architecture that combines Conditional Variational Autoencoder (CVAE) and Conditional Latent Regressor (CLR) models to enforce the connection between latent encoding and output in both directions jointly and achieve improved performance [89]. Pix2PixHD also synthesizes images from semantic labels, and it addresses two main issues of the existing work, namely, the difficulty of generating high-resolution images with GANs and the lack of details and realistic textures in previous high-resolution results [77]. Table 2 shows the performance comparison of different GAN architectures. Among the GAN candidates, we did not find an advantage in using a variational method. This implies that the random samples from a learned distribution may counteract the benefits of uncertainty modeling, which introduces higher noise to the output [51]. Our method outperforms Pix2PixHD in both application scenarios with 0.8% improvement in accuracy and 4–22 ms less inference latency.

5.4 Impact of Rollback

A key component of our system is the rollback module. Figure 9 summarizes the impact of the rollback module for different values of the user-defined parameter $x$, which controls how much accuracy...
loss can be tolerated. The trend is as expected, i.e., accuracy degrades as \( x \) increases, but we see an inflection point around \( x > 1\% \) in speech recognition and \( x > 2\% \) in event detection at which point the accuracy degrades more rapidly. Therefore, we select \( x = 1\% \) as the best trade-off between accuracy and latency for these datasets. We note that \( x \) does not need to be pre-determined and can be set adaptively depending on network conditions. For example, if the bandwidth is low (e.g., < 200 Kbps), we can set a high \( x \) to further reduce the latency, whereas if the bandwidth is high (e.g., > 800 Kbps), we can set a low \( x \) to improve the accuracy.

### 5.5 Impact of Data Alignment

We now investigate the impact of our data alignment method on accuracy and latency. Figure 10 shows the data alignment deviation between our method and ground truth. Overall, the mean deviation is 45 ms and 42 ms for speech recognition and event detection, respectively. Since we search for the minimal L1 distance between feature maps to align data streams, the granularity is one frame (e.g., 33 ms for 30 FPS or 40 ms for 25 FPS). Thus, our data alignment module achieves an average deviation of 1 to 1.5 frames. We observe that the data alignment deviation for speech recognition is higher than that for event detection. Due to the nature of lipreading images, the difference between adjacent images is smaller than that for events like car racing, which makes it hard for the data alignment module to search for the minimum L1 loss. We also investigate the accuracy and the latency differences between our system and one with perfect data alignment. We observe that the differences are marginal, and the latency added by data alignment is less than 10 ms, and the accuracy difference is nearly zero.

### 5.6 Training and Resource Overhead

Our system introduces training overhead since we train over several configurations (data resolution and delays). However, even though we train several GAN models, we do not need to use the entire training dataset since we are imputing intermediate features rather than raw sensor data. Figure 11 shows the impact of training data size on the performance of our system. By leveraging a fifth or even a tenth of the original training data, we can significantly reduce the entire training overhead down to about 8–20 hours while still retaining high accuracy. The runtime resource overhead for our system is also small (about 100 MB of memory in the GPU) because we only need to load two GANs (data imputation and rollback) into memory at one time.

### 5.7 Impact of IoT nodes

We now look at end-to-end improvement using real IoT devices and Wi-Fi radio. So far, we have assumed that the sensor data can be instantly processed on IoT devices, but in practice, there are many real-world delays, including encoding and compression on the IoT device and processing delays in the cloud. In this experiment, IoT nodes process video in real time. Figures 12(a) and 12(c) illustrate our system’s performance compared to the naive blocking method. Overall, our method still outperforms the blocking baseline under all bandwidth conditions and achieves up to 9x and 122x latency speedup on the LRW and the AVE dataset, which is consistent with our previous results. The mean computation overhead of video and audio data streaming is 4ms per frame and 3ms per second, respectively. Figures 12(b) and 12(d) show the speedup ratio for different user-defined parameters. When \( x = 4\% \), our system achieves 28x and 424x latency speedup on the LRW and AVE datasets. The results validate our previous evaluation and demonstrate that our approach offers significant benefits in real-world systems.

### 6 RELATED WORK

**Minimizing inference lag**

Existing work to minimize inference lag in deep learning has focused largely on server-side constraints, with proposed solutions seeking to optimize the utilization of available compute resources. General-purpose job schedulers [33, 71] are not well suited to lag-sensitive applications because they make scheduling decisions based on available cluster resources, prioritizing job fairness, rather than on the characteristics of jobs.

Highly specialized schedulers [29, 30, 39, 84] use pipeline profiling to determine how best to allocate resources to video analytics processes whose accuracy and lag requirements vary and whose demands may change with time. Adaptive schedulers [2, 11, 19, 25,
Compensating for misaligned sensor streams The need to align multi-modal sensor data prior to use in machine learning or deep learning applications is not new, but most existing work assumes either that 1) the sensor readings are already temporally aligned, or 2) that the raw signals need to be temporally aligned before being input to the model.

Commonly, the sensors used to record multi-modal streams are co-located on the same device (e.g., camera and microphone, or cameras and LIDAR on an autonomous vehicle [12, 15, 47, 48, 82]) and share a common clock. In other cases the sensor may be recorded by separate devices but still share a common clock that is either internal to one or external to both, or the devices will synchronize via a shared synchronization protocol [49]. In either of these scenarios, the first assumption is sound and no further processing to align the signals is required.

The second assumption applies when sensors are located on devices that are not tightly synced to a shared clock. Prior to the rise of deep learning, analytical techniques like dynamic time warping [1, 41, 86], manifold warping [26, 73] and canonical correlation analysis [31, 65, 87, 88] were used to align signals with a shared dimensionality. Deep learning techniques have been developed to compensate for the inherent dimensionality limitations of these techniques [4, 34, 69, 70], but in order to work, raw data from all sensor modalities must be received prior to alignment.

Our technique is robust to completely missing or poorly aligned data, without requiring explicit alignment of the raw signals prior to fusion. By imputing the features of delayed signals at multiple latency speedup with the same accuracy as six existing solutions.

7 CONCLUSION

We present speculative inference on multi-modal data streams to adapt to asymmetric bandwidth. The system adaptively and speculatively generates inference from multi-modal models without waiting for all data streams to be available and aligned. We implement the system on three multi-modal scenarios. Experiments on publicly available datasets show that our system achieves $7 \sim 128x$ latency speedup with the same accuracy as six existing solutions.

ACKNOWLEDGEMENTS

We sincerely thank reviewers for their insightful feedback. The research reported in this paper was sponsored in part by the CCDC Army Research Laboratory (ARL) under Cooperative Agreement W911NF-17-2-0196 (ARL IoBT CRA) and by National Science Foundation under Grant No. 1815347. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the ARL, NSF or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.