This paper has been accepted for publication in the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM) 2026, where it received the Best Paper Award. This is the author's accepted manuscript and is not the final published version. The final version of record will be available through IEEE Xplore after publication.

Multi-TAB: Multi-View Inference at the Edge with Resource-Aware Split Computing

Tanzil B. Hassan*, Kevin S. Chan†, Fikadu Dagefu†, Jonathan Ashdown§, Flavio Esposito¶, Francesco Restuccia*
*Northeastern University, USA; †DEVCOM Army Research Laboratory, USA;
§United States Air Force, USA; ¶Saint Louis University, USA

***Abstract*—Multi-view collaborative inference through deep neural networks (DNNs) aggregates views from mobile devices, thus achieving holistic situational awareness in applications such as surveillance and autonomous driving. Since on-device execution incurs significant resource consumption, offloading DNN computation to edge servers can alleviate latency demands. To this end, split computing partitions DNNs across devices and transmits intermediate compressed features to decrease the networking load. Nevertheless, unreliable wireless links between mobile devices and edge servers can increase end-to-end latency during offloading. Accordingly, we propose a novel multi-view inference framework named `Multi-TAB` to optimize the trade-off between DNN performance and networking resource consumption. In contrast to prior work, `Multi-TAB` supports split computing while adaptively compressing the intermediate latent representations at runtime based on current capacity of the wireless channel. We demonstrate the effectiveness of our framework on two challenging pedestrian density detection tasks on the WILDTRACK and MULTIVIEWX datasets. We prototyped `Multi-TAB` on the Colosseum network emulator and evaluated its latency and overhead data volume on Jetson Nano and Raspberry Pi 4. Experimental results show that `Multi-TAB` can reduce end-to-end inference latency by 33.7% compared to state-of-the-art approaches while maintaining accuracy loss within 2.4%.**

I. Introduction

Many applications such as smart drones, robotics, and security camera systems [1] require cooperative inference tasks based on computer vision to implement person identification [2], pedestrian density measurement [3], multi-view object detection [4] and target tracking [5], among others. The key issue is that the volume of the collected data (e.g., images, high-definition videos, and point clouds) may lead to excessive communication overhead. Multi-view inference multiplies the networking requirements through simultaneous streaming from several cameras. Excessive packet queues or retransmissions force late frames to be dropped (degrading task accuracy) or stall processing (increasing latency). Balancing this trade-off is therefore paramount in multi-view edge computing [6].

Existing Issues. Most existing multi-view methods [3, 4, 7–11] assume that rich deep features are readily available, either by running heavy backbones on every device or by offloading substantial intermediate data, and therefore do not address on-device resource constraints. For instance, multi-view image classification can tolerate loosely correlated views, whereas person re-identification requires tightly aligned, highly informative features from every camera [12]. In practice, edge devices are often *resource-constrained* (e.g., low-power embedded boards), making it infeasible to run deep feature extractors locally. Split computing [13] offers a partial remedy by offloading deeper layers to an edge server, but early-layer features are generic and not task-aligned; aggressive compression of such features increases task error and makes reconstruction difficult.

Task-oriented multi-view/multi-device inference [14] was proposed to reduce transmission cost and manage latency, but these methods are difficult to deploy at the edge because their feature extraction modules are parameter-heavy, especially as the number of devices or task-related complexity grows [9]. Figure 1 illustrates the key challenges in a multi-view edge inference system. None of these approaches directly solves the central dilemma in low-power multi-view edge inference: *how to extract and transmit only the most task-relevant information under strict compute and bandwidth limits, without sacrificing downstream performance*.

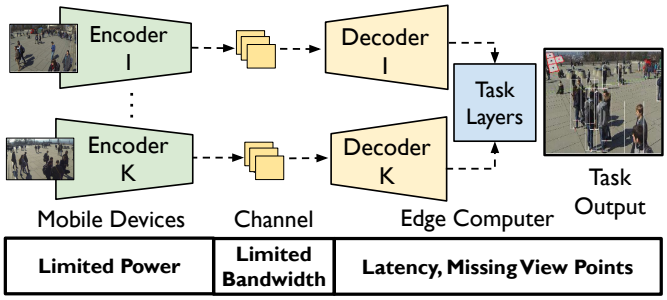


Figure 1: Challenges of Multi-view Inference at the Edge.

Proposed Approach. To address the above challenges, we propose `Multi-TAB` to enable split computing for multi-view edge inference. Split computing partitions a DNN between the device and the edge server, running the early layers locally and off-loading the deeper layers. Specifically, we take a pretrained model (trained on a large dataset) and split it into a head and a tail deep neural network (DNN). The head part is deployed on mobile devices and the tail part on the edge server. We apply a light-weight encoder-decoder architecture to encode task-oriented distributed features.

In contrast to existing split-computing work [13], the multi-view scenario imposes additional unique constraints, namely asynchronous latency across participating mobile devices during inference. To address these issues, we design a new information-theoretic framework to train the encoder-decoder structure to operate with *task-oriented* feature-maps under a

tight resource budget. As such, Multi-TAB enables selective transmission that adapts to missing camera views and unfavorable wireless conditions, thereby reducing data transmission without compromising task performance. In addition, we employ a lightweight, wireless channel context-aware masking policy that evaluates every feature map using two criteria — how valuable the view is to the task and the current wireless channel condition. Based on this, we apply sparsification on the feature map to reduce transmission data size. By combining information-bottleneck compression with adaptive sparsification, Multi-TAB delivers a better accuracy-versus-bandwidth trade-off with respect to existing state of the art [15–19].

Summary of Novel Contributions

- We propose Multi-Head Task-Adaptive Bottleneck (Multi-TAB), a novel multi-view inference framework that splits a pretrained multi-view model into edge-side lightweight "head" and server-side "tail" networks with an encoder–decoder in between, so that intermediate feature maps are compressed on-the-fly into task-relevant, view-prioritized representations. Unlike prior work that hard-codes split points or relies on task-agnostic tensor compression, Multi-TAB requires no extensive architectural modification. Guided by a distributed information-bottleneck objective, Multi-TAB produces task-relevant multi-view latent features, while an SNR-conditioned masking mechanism induces structured sparsity to prune redundant feature maps.

  Together, these components dynamically balance bandwidth, latency, and accuracy by minimizing mutual information while preserving task performance, thereby bounding transmission redundancy and enabling communication-efficient multi-view inference under fluctuating wireless conditions.

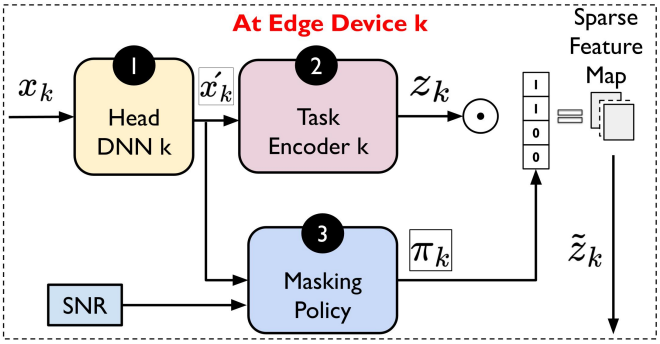
- We have tested Multi-TAB on the Colosseum [20] digital-twin network emulator and WiFi based testbed. We benchmarked resource consumption performance on Jetson Nano and Raspberry Pi 4 and execute the tail on a workstation. We assess two multi-view vision datasets based on pedestrian-density estimation task called the WILDTRACK and MULTI-VIEWX [3] datasets. For each backbone architecture (ResNet-18 and ResNet-50), we measure and report the task accuracy, per-frame end-to-end latency, uplink overhead transmission, and computation on the device. These results are compared with those of state-of-the-art multi-view inference baselines [15–19]. Experimental results show that Multi-TAB reduces end-to-end inference latency by 33.7% compared to state-of-the-art methods, while keeping accuracy loss within 2.4%.

II. System Model and Preliminaries

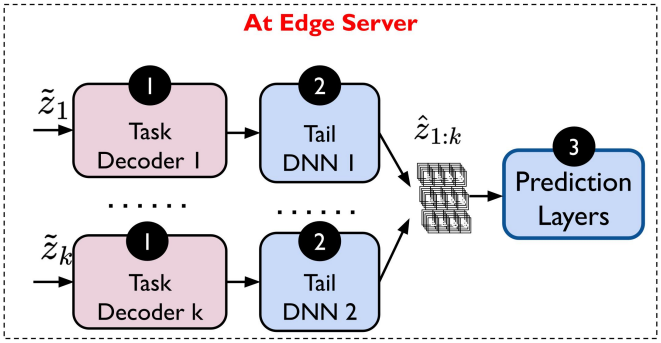
In this section, we walk through the Multi-TAB system model. Multi-TAB allows splitting a convolutional neural network (CNN) model and deploying it distributively across edge devices and the server.

Figure 2 details the Multi-TAB inference pipeline, separated into device-side process in Figure 2a and server-side process in Figure 2b. The process begins on the mobile device (Figure 2a) with **(Step 1)**, where the backbone DNN is partitioned into a lightweight head, and a server-side tail. The head processes raw input $\boldsymbol{x}_k$ to extract latent features $\acute{x}_k$. This is followed by **(Step 2)**, the task encoder trained to generate task-oriented latent representations $\boldsymbol{z}_k$. In **(Step 3)**, a masking policy uses the pooled representation from $\acute{x}_k$ and the current Signal-to-Noise Ratio (SNR) to generate a binary mask π_k to be applied on the task-oriented feature z_k and convert it into sparse feature-maps $\tilde{\boldsymbol{z}}_k$. Then $\tilde{\boldsymbol{z}}_k$ is transmitted after encoding according to the communication protocol.

The server-side process Figure 2b handles reconstruction and fusion. In **(Step 1)**, the incoming separate bitstreams from k are received and decoded by the task decoder to reconstruct the sparse representations from the k devices. Since edge devices are distributed across different locations, their feature SNR profiles vary. Next, in **(Step 2)**, the reconstructed features are forwarded to and processed by the tail DNN. Finally, in **(Step 3)**, the task layers concatenate the outputs to form the combined feature set $\hat{z}_{1:k}$, which is fed into the prediction layer to perform the downstream inference task.



(a) Edge device deployment components



(b) Edge server deployment components

Figure 2: Multi-TAB Inference Model. (a) The feature extraction steps at edge devices. (b) The edge server processing steps.

A. Probabilistic Modeling

Random variables are denoted by upper-case letters (e.g., X, Y) and their specific realizations by bold lower-case letters (e.g., $\boldsymbol{x}, \boldsymbol{y}$). A collection of N random variables $\{X_1, \ldots, X_N\}$ is abbreviated as $X_{1:N}$. We use $H(Y)$, $H(Y|X)$, and $H(X, Y)$ to represent the entropy of Y, the

conditional entropy of Y given X, and the joint entropy of (X, Y), respectively. The mutual information between X and Y is defined as $I(X;Y)$. For distributions $p(\boldsymbol{x})$ and $q(\boldsymbol{x})$, their Kullback–Leibler (KL) divergence is $D_{\mathrm{KL}}(p\|q)$ and their cross-entropy is $H(p,q)$. The expectation of a random variable X is denoted by $\mathbb{E}[X]$. A Gaussian distribution with mean vector μ and variance σ is expressed as $\mathcal{N}(\mu, diag(\sigma^2))$. A uniform distribution over an interval (a,b) is denoted by $\mathcal{U}(a,b)$. We indicate quantization as $\lfloor \boldsymbol{x} \rceil$.

Multi-device cooperative inference involves K mobile devices and an edge server. The input views of different mobile devices, denoted as $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_K)$, and the target variable $\boldsymbol{y}$, are denoted as different realizations of the random variables $(X_1, \ldots, X_K, Y)$. The data from $\boldsymbol{x_k}$ and $\boldsymbol{y}$ can be sampled from the joint distribution $p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_K, \boldsymbol{y})$. Let us consider a latent representation of features $\boldsymbol{z}_k$ containing only task-relevant information extracted from input variable $\boldsymbol{x}_k$, which is further encoded into vector $\boldsymbol{v}_k$ for efficient wireless transmission to the edge server. After receiving the encoded features from the mobile devices, the edge server performs further processing to derive inference result. The extracted features $(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_K)$ are instantiated by random variables $(Z_1, \ldots, Z_K)$. Thus, we define a Markov chain comprising these random variables as follows:

$$Y \leftrightarrow X_k \leftrightarrow Z_k \ \forall k \in \{1, \ldots, K\}, \tag{1}$$

Equation (1) satisfies the conditional probability distribution $p(\boldsymbol{z}_k, \boldsymbol{x}_k|\boldsymbol{y}) = p(\boldsymbol{z}_k|\boldsymbol{x}_k)p(\boldsymbol{x}_k|\boldsymbol{y})$, $\forall k \in \{1, \ldots, K\}$. The high dimensionality of the input X_k produces information redundancy – for example, background, irrelevant objects, etc. – that is not relevant to correctly predict Y. As such, the latent representation Z_k should capture only the task related information from the observation variable X_k, according to system constraints. Next, we detail the proposed distributed feature extraction.

III. The Multi-TAB Inference Framework

The prediction performance of Multi-TAB is mostly related to the quality of the extracted features. However, compressing the information degrades the quality of the extracted features. As a result, we pursue a communication-performance tradeoff. From an information-theoretic standpoint, Multi-TAB aims at generating feature Z_k from K devices as a maximally compressed version of input X while being maximally predictive of output variable Y. We leverage the Information Bottleneck (IB) [21], whose objective is formulated as follows:

$$\begin{aligned}\mathcal{L}_{IB}(\beta) &= -I(Y;Z_k) + \beta I(X_k;Z_k) \\ &= -H(Y|Z_k) + H(Y) + \beta I(X_k;Z_k) \\ &= \mathbb{E}\Big[\ \mathbb{E}[-\log p(\boldsymbol{y}|\boldsymbol{z_k})] + \beta\ D_{KL}(p(\boldsymbol{z_k}|\boldsymbol{x_k})||p(\boldsymbol{z_k}))\Big]\end{aligned} \tag{2}$$

where $\beta \geq 0$ is a weighting factor controlling the communication and performance tradeoff. The term $H(Y)$ can be eliminated as it is constant. Since minimizing Equation 2 requires obtaining the marginal distribution of Z_k, which is computationally expensive, we adopt the variational method [22] to reformulate the upper-bound of the objective.

A. Variational upper-bound formulation

For each device k, we define $p_{\boldsymbol{\theta}_k}(\boldsymbol{z}_k|\boldsymbol{x}_k)$ as the conditional distribution parameterized by $\boldsymbol{\theta}_k$. The function $f_{\boldsymbol{\theta_k}}(.)$ encodes the input $\boldsymbol{x}_k$ to latent representation $\boldsymbol{z}_k$. Given the joint distribution $p(\boldsymbol{x}_k, \boldsymbol{y})$ and the Markov chain in (1), the marginal distribution $p(\boldsymbol{z}_k)$ and the conditional distribution $p(\boldsymbol{y}|\boldsymbol{z})$ are fully determined by the conditional distribution $p_{\boldsymbol{\theta}_k}(\boldsymbol{z}_k|\boldsymbol{x}_k)$ as follows:

$$\begin{aligned}p(\boldsymbol{z}_k) &= \int p(\boldsymbol{x}_k) p_{\boldsymbol{\theta}_k}(\boldsymbol{z}_k|\boldsymbol{x}_k) d\boldsymbol{x}_k, \\ p(\boldsymbol{y}|\boldsymbol{z}_k) &= \int \frac{p(\boldsymbol{x}_k, \boldsymbol{y}) p_{\boldsymbol{\theta}_k}(\boldsymbol{z}_k|\boldsymbol{x}_k)}{p(\boldsymbol{z}_k)} d\boldsymbol{x}_k,\end{aligned}$$

Since the integrals above are computationally expensive, we approximate them using variational approximation [23]. Specifically, we introduce two probability distributions $q_k(\boldsymbol{z}_k)$ and $q_{\boldsymbol{\phi_k}}(\boldsymbol{y}|\boldsymbol{z}_k)$ as the variational distributions to approximate $p(\boldsymbol{z}_k)$ and $p_{\boldsymbol{\theta_k}}(\boldsymbol{y}|\boldsymbol{z}_k)$, respectively. The first term of Equation (2) is

$$\begin{aligned}&\mathbb{E}_{p(\boldsymbol{x}_k,\boldsymbol{y})}\Big[\mathbb{E}_{p_{\theta_k}(\boldsymbol{z}_k|\boldsymbol{x}_k)}\left[-\log p(\boldsymbol{y} \mid \boldsymbol{z}_k)\right]\Big] \\ &= \mathbb{E}_{p(\boldsymbol{x}_k,\boldsymbol{y})}\Big[\mathbb{E}_{p_{\theta_k}(\boldsymbol{z}_k|\boldsymbol{x}_k)}\left[-\log q_{\phi_k}(\boldsymbol{y} \mid \boldsymbol{z}_k)\right]\Big] \\ &\quad - \mathbb{E}_{p(\boldsymbol{x}_k,\boldsymbol{y})}\left[\mathbb{E}_{p_{\theta_k}(\boldsymbol{z}_k|\boldsymbol{x}_k)}\left[\log \frac{p(\boldsymbol{y} \mid \boldsymbol{z}_k)}{q_{\phi_k}(\boldsymbol{y} \mid \boldsymbol{z}_k)}\right]\right] \\ &\leq \mathbb{E}_{p(\boldsymbol{x}_k,\boldsymbol{y})}\Big[\mathbb{E}_{p_{\theta_k}(\boldsymbol{z}_k|\boldsymbol{x}_k)}\left[-\log q_{\phi_k}(\boldsymbol{y} \mid \boldsymbol{z}_k)\right]\Big]\end{aligned}$$

which is the upper-bound of task performance. The upper-bound of the next term of Equation 2 is

$$\begin{aligned}&D_{\mathrm{KL}}\Big(p_{\theta_k}(\boldsymbol{z}_k \,|\, \boldsymbol{x}_k) \,\|\, p_{\theta_k}(\boldsymbol{z}_k)\Big) \qquad (3)\\ &= D_{\mathrm{KL}}\Big(p_{\theta_k}(\boldsymbol{z}_k \,|\, \boldsymbol{x}_k) \,\|\, q_{\phi_k}(\boldsymbol{z}_k)\Big) \\ &\quad - D_{\mathrm{KL}}\Big(p_{\theta_k}(\boldsymbol{z}_k) \,\|\, q_{\phi_k}(\boldsymbol{z}_k)\Big) \\ &\leq D_{\mathrm{KL}}\Big(p_{\theta_k}(\boldsymbol{z}_k \,|\, \boldsymbol{x}_k) \,\|\, q_{\phi_k}(\boldsymbol{z}_k)\Big). \qquad (4)\end{aligned}$$

where the term $D_{\mathrm{KL}}\big(p_{\theta_k}(z_k) \,\|\, q_{\phi_k}(z)\big)$ is ≥ 0. The family of variational conditional distributions is parameterized by $\boldsymbol{\phi}_k$. With these approximations, we re-write the VIB objective as

$$\begin{aligned}\mathcal{L}_{\mathrm{VIB},k}(\beta; \boldsymbol{\theta}_k, \boldsymbol{\phi}_k) = \mathbb{E}\Big\{&\mathbb{E}\big[-\log q_{\boldsymbol{\phi_k}}(\boldsymbol{y} \mid \boldsymbol{z}_k)\big] \\ &+ \beta\, D_{\mathrm{KL}}\big(p_{\boldsymbol{\theta_k}}(\boldsymbol{z_k} \mid \boldsymbol{x_k}) \,\|\, q_{\phi_k}(\boldsymbol{z}_k)\big)\Big\}.\end{aligned} \tag{5}$$

B. Training the VIB Objective with DNN.

The DNN-based training of the VIB objective in Equation 5 is summarized in Algorithm 1. We model the encoder distribution $p_{\boldsymbol{\theta}_k}(\boldsymbol{z}_k|\boldsymbol{x}_k)$ as a multivariate Gaussian, where both the mean vector $\boldsymbol{\mu}_k \in \mathbb{R}^d$ and the standard deviation vector $\boldsymbol{\sigma}_k \in \mathbb{R}^d$ are predicted by a DNN parameterized by $\boldsymbol{\theta}_k$. The covariance matrix is assumed diagonal, i.e.,

$\mathrm{diag}\{\sigma_{k,1}^2, \ldots, \sigma_{k,d}^2\}$. The decoder distribution $p_{\boldsymbol{\phi}_k}(\boldsymbol{y}|\boldsymbol{z}_k)$ is also parameterized by a DNN with parameters $\boldsymbol{\phi}_k$. As a regularizer, the variational marginal $q_{\boldsymbol{\phi}_k}(\boldsymbol{z}_k)$ is set as an isotropic standard Gaussian, $\mathcal{N}(\boldsymbol{z}_k|\boldsymbol{0}, \boldsymbol{I})$. These notations are consistently used in both the objective and Algorithm 1.

To enable end-to-end training, we leverage the reparameterization trick [24]: for each sample, the latent code is generated as $\boldsymbol{z}_k = \boldsymbol{\mu}_k + \boldsymbol{\sigma}_k \odot \boldsymbol{\epsilon}_k$, where $\boldsymbol{\epsilon}_k$ is sampled from $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. The KL-divergence term in Equation (5) simplifies to $\sum_{i=1}^{d} \left\{ \frac{\mu_{k,i}^2 + \sigma_{k,i}^2 - 1}{2} - \log \sigma_{k,i} \right\}$. The conditional likelihood $p_{\boldsymbol{\phi}_k}(\boldsymbol{y}|\boldsymbol{z}_k)$ is formulated based on the output of the decoder network and a task loss function $\ell(\boldsymbol{y}, \hat{\boldsymbol{y}}_k)$, with $p_{\boldsymbol{\phi}_k}(\boldsymbol{y}|\boldsymbol{z}_k) \propto \exp(-\ell(\boldsymbol{y}, \hat{\boldsymbol{y}}_k))$. Monte Carlo sampling is used to estimate the objective and its gradients, allowing stochastic optimization with respect to both encoder and decoder parameters.

$$\mathcal{L}_{\mathrm{VIB},k}(\beta; \boldsymbol{\theta}_k, \boldsymbol{\phi}_k) \simeq \frac{1}{M} \sum_{m=1}^{M} \{ -\log p_{\boldsymbol{\phi}_k}(\boldsymbol{y}^m|\boldsymbol{z}_k^m) + \beta \sum_{i=1}^{d} \left\{ \frac{\mu_{k,i}^2 + \sigma_{k,i}^2 - 1}{2} - \log \sigma_{k,i} \right\} \Bigg\}. \tag{6}$$

Algorithm 1 proceeds as follows. The training dataset $\mathcal{D}$, batch size M, and initial model parameters $\boldsymbol{\theta}_k$ and $\boldsymbol{\phi}_k$ are initialized (**line 1**). At each iteration, a mini-batch of samples $\{(\boldsymbol{x}_k^m, \boldsymbol{y}^m)\}_{m=1}^M$ is drawn from $\mathcal{D}$ (**line 2**). For each mini-batch sample, the encoder network outputs the corresponding mean $\boldsymbol{\mu}_k^m$ and standard deviation $\boldsymbol{\sigma}_k^m$ (**line 3**). Standard Gaussian noise $\boldsymbol{\epsilon}_k^m$ is sampled (**line 4**), and latent codes $\boldsymbol{z}_k^m$ are constructed using the reparameterization trick (**line 5**). The VIB loss $\mathcal{L}_{\mathrm{VIB},k}$, as described in Equation (6), is computed for the batch (**line 6**). The model parameters are updated via stochastic gradient descent (**line 7**). Training continues until the validation loss fails to improve for a set number of epochs, which serves as the convergence criterion (**line 8 and line 9**).

Algorithm 1 Training of Task-Relevant Feature Extractor

Require: Dataset $\mathcal{D}$, small batch size M, initialized parameters $\boldsymbol{\theta}_k$, $\boldsymbol{\phi}_k$.
Ensure: The optimized parameters $\boldsymbol{\theta}_k$ and $\boldsymbol{\phi}_k$
1: **repeat**
2: Sample a minibatch $\{(\boldsymbol{x}_{1:K}^m \boldsymbol{y}^m)\}_{m=1}^M$ from $\mathcal{D}$.
3: Compute $\{\boldsymbol{\mu}_k^m\}_{m=1}^M$ and $\{\boldsymbol{\sigma}_k^m\}_{m=1}^M$.
4: **for** $m = 1$ to M **do**
5: Sample $\boldsymbol{\epsilon}_{\boldsymbol{k}}^{\boldsymbol{m}} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$.
6: Compute $\boldsymbol{z}_k^m = \boldsymbol{\mu}_k^m + \boldsymbol{\sigma}_k^m \odot \boldsymbol{\epsilon}_{\boldsymbol{k}}^{\boldsymbol{m}}$.
7: Compute the loss $\mathcal{L}_{\mathrm{VIB},k}$ based on (6).
8: Update parameters $\boldsymbol{\theta}_k, \boldsymbol{\phi}_k$
9: **until** Convergence $\boldsymbol{\theta}_k$ and $\boldsymbol{\phi}_k$

C. Feature Map Compression with masking.

After executing Algorithm 1, `Multi-TAB` learns to extract task-relevant latent representations $\boldsymbol{z}_k$ from input features $\boldsymbol{x}_k$ to minimize the mutual information $I(X_k; Z_k)$, while preserving predictive power for label $\boldsymbol{y}$. Although these representations are already information-efficient in the variational sense, transmitting them over a constrained communication link still requires further encoding. To enable efficient transmission, we quantize and encode the latent features into a compact bitstream. Following the entropy bottleneck framework [19], the quantized features are passed through an entropy bottleneck model, which estimates the probability distribution over quantized values and encodes them into a variable-length binary stream using entropy coding. While this encoding is a quantized tensor, the number of bits required to encode depends on the value of each element in the tensor. Therefore, to reduce the data size, when the wireless channel condition is poor, we introduce an adaptive sparsity mechanism.

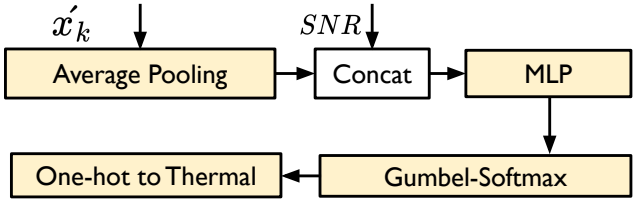


Figure 3: Masking policy workflow

Algorithm 2 SNR-Conditioned Masking

Require: Feature map $\acute{\boldsymbol{x}}_k \in \mathbb{R}^{C \times H \times W}$, task-oriented feature $\boldsymbol{z}_k$, SNR γ, candidate set $\mathcal{R} = \{r_1 < \cdots < r_L\}$, temperature τ.
Ensure: Contiguous mask $\boldsymbol{\pi} \in \{0,1\}^C$ and masked feature map $\tilde{\boldsymbol{z}}_k$.
$\boldsymbol{s} \leftarrow \mathrm{AvgPool}(\acute{\boldsymbol{x}}_k) \in \mathbb{R}^C$
$\boldsymbol{v} \leftarrow [\boldsymbol{s}; \gamma]$
$\boldsymbol{a} \leftarrow \mathrm{MLP}_{\text{policy}}(\boldsymbol{v}) \in \mathbb{R}^L$ ▷ logits over $\mathcal{R}$
if Training **then**
  $\hat{\boldsymbol{v}} \leftarrow \mathrm{GumbelSoftmax}(\boldsymbol{a}, \tau) \in \mathbb{R}^L$
  $\bar{\boldsymbol{v}} \leftarrow \mathrm{OneHot}(\arg\max_\ell \hat{v}_\ell) \in \{0,1\}^L$
else
  $\ell^\star \leftarrow \arg\max_\ell a_\ell$
  $\bar{\boldsymbol{v}} \leftarrow \mathrm{OneHot}(\ell^\star) \in \{0,1\}^L$
for $\ell = 1$ to L **do**
  $t_\ell \leftarrow \sum_{j=\ell}^{L} \bar{v}_j$
$\hat{R} \leftarrow \sum_{\ell=1}^{L} \bar{v}_\ell \, r_\ell$
$r^\star \leftarrow \min\{r \in \mathcal{R} : \; r \geq \hat{R}\}$
$\pi_c \leftarrow 0, \; \forall c \in \{1, \ldots, C\}$
for $c = 1$ to $r^\star$ **do**
  $\pi_c \leftarrow 1$
$\tilde{\boldsymbol{z}}_k \leftarrow \boldsymbol{z}_k \odot \boldsymbol{\pi}$
return $\boldsymbol{\pi}, \tilde{\boldsymbol{z}}_k$

Policy mask generation. The structure of the policy mask is illustrated in Figure 3 and summarized in Algorithm 2. First, the input feature map $\acute{\boldsymbol{x}}_k$ is average-pooled over the spatial dimensions to obtain global information on the feature map. As the encoder is already trained on task-oriented features we

get information from $\boldsymbol{s} \in \mathbb{R}^C$ **(line 1)**, which is concatenated with the perceived SNR value γ to form a context vector $\boldsymbol{v}$ **(line 2)**. This context is passed through a lightweight 2-layer policy MLP (followed by softmax implicitly) to produce logits $\boldsymbol{a} \in \mathbb{R}^L$ over the discrete candidate set $\mathcal{R}$ **(line 3)**. The set $\mathcal{R}$ can represent the number of channels to reduce within this limit. During training, a discrete decision is sampled via Gumbel-Softmax with temperature τ to obtain a soft one-hot vector $\hat{\boldsymbol{v}}$ **(line 5)**, which is then converted to a hard one-hot selection $\bar{\boldsymbol{v}}$ using $\arg\max$ **(line 6)**, while at inference the selection is obtained directly from $\arg\max$ over logits **(lines 8–9)**. Next, the one-hot decision is transformed into a thermometer-encoded vector through a cumulative sum **(lines 10–11)**, which enforces resulting channel mask always corresponds to contiguous prefix channels of the form $[1, \cdots, 1, 0, \cdots, 0]$. The selected channels are computed as $\hat{R} = \sum_{\ell=1}^{L} \bar{v}_\ell r_\ell$ **(line 12)** and snapped to the nearest feasible option in $\mathcal{R}$ via a conservative rounding **(line 13)**, after which the contiguous binary mask $\boldsymbol{\pi} \in \{0,1\}^C$ is constructed by activating the first $r^\star$ channels **(lines 14–16)** and applied to the feature map via elementwise multiplication to produce $\tilde{\boldsymbol{z}}_k = \boldsymbol{z}_k \odot \boldsymbol{\pi}$ **(line 17)**, returning both $\boldsymbol{\pi}$ and $\tilde{\boldsymbol{z}}_k$ for subsequent transmission and decoding **(line 18)**. We sample the decision as a one-hot vector using Gumbel-Softmax [25] and transform it into thermometer encoding of vector $\boldsymbol{\pi}$; this ensures that contiguous feature-map channels are always pruned from the input.

Loss function. During training, we minimize the following objective function to encourage improving the task accuracy along with minimizing the number of active feature map channels to reduce the bandwidth usage. The first term of the loss is the task based loss (i.e, classification error, bounding box error) and second term encourages minimizing the number of feature maps in the output of the transmitted $\tilde{\boldsymbol{z}}_k$ feature.

$$\mathcal{L}_{\text{masking},k}(\lambda; \boldsymbol{\phi}_k) = \mathbb{E}\left[-\log q_{\boldsymbol{\phi}_k}(\boldsymbol{y} \mid \tilde{\boldsymbol{z}}_k) + \lambda \sum_{c=1}^{C} \pi_c \right]. \quad (7)$$

In Equation 7, the term λ tunes the penalty for over using the communication bandwidth and task accuracy, detailed in the Section IV.

D. Server side models and training

At the server side, the received sparse feature maps are decoded from the bitstream and zero-padded for the encoder side of the split model. The task decoder extracts the task-related features from $\tilde{\boldsymbol{z}}_k$ and passes them through the tail side of the split DNN. For each device, we generate a concatenation of combined features $z'_{1:k}$ to be the input of the task layers. These task layers vary according to the architecture of the model and the task-solving algorithm. All that is required is features extracted from a large DNN, so that decisions based on those feature maps become more accurate. `Multi-TAB` generates these features by splitting the large feature extractor into a head-tail pair to reduce computation at the edge devices.

Training is a two stage process, first we train the task encoder and decoder with Algorithm 1. After that we freeze the encoder parameters to continue with Algorithm 2 to train the masking policy. We consider large τ for Gumbel-softmax initial epochs and later anneal it down to 0.20 to make the softmax decision harder at each iteration.

IV. Experimental Evaluation

We extensively evaluate `Multi-TAB` with a set of experiments: (1) end-to-end simulations to test the robustness of `Multi-TAB` against multiple SNR profiles; (2) using the Colosseum wireless emulator with two scenarios containing fixed path loss of 5 dB and 30 dB respectively. More in detail in IV-B; (3) using an experimental testbed comprising of Jetson Nano and Raspberry Pi 4 connected to an edge server using Wi-Fi to experimentally benchmark the end-to-end offloading latency.

A. Datasets, Models and Metrics

Datasets and Metrics. We evaluate multi-camera pedestrian occupancy prediction task on the WILDTRACK and MULTIVIEWX [3] datasets. WILDTRACK provides 400 synchronized frames from 7 cameras (each 1080×1920 pixels), covering a $12\,\text{m} \times 36\,\text{m}$ area. Each frame contains an average of 20 persons. MULTIVIEWX [3], a synthetic dataset, covers a $16\,\text{m} \times 25\,\text{m}$ region, with 6 overlapping cameras (also 1080×1920). MULTIVIEWX is more challenging, averaging 40 persons per frame. For both datasets, we adopt MVDet [26] as baseline model architecture for pedestrian occupancy detection. Performance is measured using the Multiple Object Detection Accuracy (MODA) metric [27]. MODA penalizes both **missed detections** (*FN*) and **false positives** (*FP*) by normalizing their sum with the total number of ground-truth objects (*GT*), namely $\text{MODA} = 1 - \frac{\text{FN}+\text{FP}}{\text{GT}}$.

Models. In `Multi-TAB`, we utilize ResNet-18 , splitting the DNN after the first stage. Most of the experiments were done at this split settings. At this split point, the feature map output is of size $64 \times 56 \times 56$ for ResNet-18. By design, a lightweight masking policy module is composed of a Global Average Pooling (GAP) layer followed by a three-layer Multi Layer Perception (MLP). For ResNet-18, the policy masking MLP takes as input a 65-dimensional vector (64 feature channels plus a SNR estimate dimension). For the bottleneck layer, we design a 2-layer convolutional 2D block to downsample the feature maps into $16 \times 23 \times 40$ dimension and upsample at the server side. This entire masking policy pipeline (including both Bottleneck layers, GAP and MLP) requires only 221,312 MACs and 24,864 parameters for ResNet-18. Compared to the total computation of the backbone models (1.8×10^9 MACs for ResNet-18), the masking policy module contributes less than 0.02% computational overhead.

B. Experimental Setup

Simulation Setup. To model a wireless transmission scenario, we segment the application layer data into fixed-size packets following the typical Maximum Transmission Unit (MTU) constraint of 1500 bytes. At each simulation time

step, the instantaneous SNR is computed based on log-distance path loss, log-normal shadowing, and co-channel interference, along with additive thermal noise modeled as Additive White Gaussian Noise (AWGN). The SNR is then mapped to an achievable channel throughput using a Shannon-based capacity approximation over a 2 MHz bandwidth. For each packet, the transmission latency is calculated as the ratio of its size to the current channel throughput. We choose this model to emulate very low channel capacity based simulation.

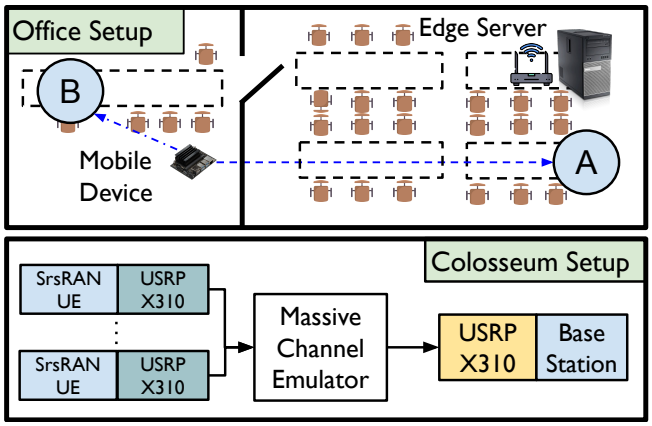


Figure 4: Experimental testbed in "Office" environment and Colosseum.

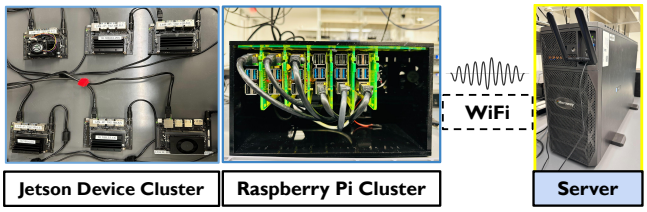


Figure 5: Emulation testbed setup.

Experimental Testbed. The experimental setup is illustrated in Figure 4. For end-to-end latency benchmarking, as mobile device, we use an NVIDIA Jetson Nano, quad-core ARM 1.9GHz CPU and mounting a 128-core GPU operating at 0.95GHz and 4GB of 64-bit LPDDR4 memory and a Raspberry Pi 4, housing a quad-core ARM 1.5GHz CPU and 2GB of LPDDR4 memory. The edge server was implemented by a Linux-based machine, equipped with an Intel(R) Xeon(R) CPU, and an NVIDIA A100 GPU. We used an office environment over-the-air Wi-Fi setup and data was collected on a single day. We use Linux `wavemon` tool to gather corresponding link quality information. As illustrated in the figure the location A - the link quality was perceived values as '70', whereas, at location 'B the link quality was measured as '43'. This setup was to measure per-device latency and overhead microbenchmark rather than a synchronized multi-device deployment. By running repeating measurements across two points for 7 views in WILDTRACK dataset, we obtain controlled compute/feature-generation costs, while multi-device end-to-end behavior is evaluated in Colosseum. In the Colosseum emulator [20], we use CPU-based Linux container to run the 5G Base Station (BS) (server) and User Equipments (UEs) (mobile devices). We emulate a scenario with 7 UEs with uplink TCP traffic [28]. Figure 5 shows the setup of our emulation testbed using two clusters one with Raspberry Pi devices and another one with Jetson Nano device with a GPU server acting as edge server. We set various limits on the Wireless Local Area Network (WLAN) interface to simulate different communication bottlenecks,

C. Baseline Methods

We compare `Multi-TAB` with *input compression*, *split inference*, and *task-oriented multi-device inference*.

Input Compression: We consider JPEG [18], and two state-of-the-art neural compression techniques, namely hyper prior **(DIC)** [19] implemented using CompressAI [29] for input image compression. For the latency benchmark, we set the JPEG quality level to 100 to represent the highest-fidelity input-compression baseline; for DIC we pick the highest quality compression that provides the highest reconstruction accuracy. We benchmark the latency for these individual image compression methods.

Split Inference. We consider Bottlefit **(BF)** [16], where a bottleneck encoder-decoder is placed after the early layer of the DNN to reduce the feature map size. We have used the first splitting point configuration as mentioned in [16] and the best bottleneck size (12 x 29 x 29 tensor). We also consider NeuroSurgeon (NS) [17], where we split after the first convolutional block of ResNet models as backbone.

Task-Oriented Communication. We consider **PIB** [15], which assigns priority based on the fixed SNR profile of the mobile devices against the edge server, by weighing each view's importance. We set the temporal parameters to zero, only to deal with each frame inference at a time. Collecting multiple frames for temporal fusion induces extra latency, which is avoided for fair comparison.

We train our framework with random initialization of all the DNN parameters. After training the model with Algorithm 1, we attach the masking policy network with the training loop. We train the feature extraction algorithm with a learning rate of maximum 0.1 with continuous annealing and masking policy with $lr = 10^{-3}$. The value of β was set to 10^{-4} and 10^{-3} in most of the experiments and $\lambda = 10^{-3}$ for experiments related to latency and overhead data size computation. During training we sample SNR uniformly from 0–25 dB during training.

D. Experimental Results

We report on communication overhead, end-to-end task processing latency, and performance on inference task. In all our experiments, we used the dataset images exactly as provided, without altering their resolution or quality.

Task processing latency. We measure the task processing latency as the sum of mobile device computing time, offloading latency, and server processing time in the WILDTRACK dataset. With the "Office" experimental case, we augment the end-to-end latency calculation by taking the average of the processing time of all the image frames in the test dataset.

We report the average latency after 100 rounds of testing the baselines. We compare the latency with distributed inference methods, additionally reporting the end-to-end latency of only image offloading (in short, OL).

Figure 6 and 7 report the end-to-end latency results, where for simplicity Multi-TAB is reported as "TAB". In terms of end-to-end latency, Multi-TAB is superior to all the baselines in all the experimental settings. The breakdown of the latency between mobile-side computing, offloading and server-side computing gives us more granular insights. In the Colosseum scenario with 5dB loss (Figure 6a) Multi-TAB improves end-to-end latency by 33.7% with respect to BF. A similar trend can be observed in other experiments. The baseline NS performs the worst due to the large intermediate feature maps. Another interesting finding is that in terms of mobile device processing, BF takes the least amount of time in all the scenarios. In Jetson Nano with WiFi, we notice that BF takes 15.09 ms in ResNet-18 (Figure 7a) compared to Multi-TAB taking 22.9 milliseconds and 17.89 milliseconds. This is because of the two-stage knowledge distillation in BF backbone, which compresses the backbone to a smaller-sized model. However, Multi-TAB reduces communication latency by creating encoded feature maps. Importantly, Multi-TAB increases latency for WiFi position A to B 43.24% and in Colosseum from 5dB path loss to 30 dB path loss by only 14.73%.

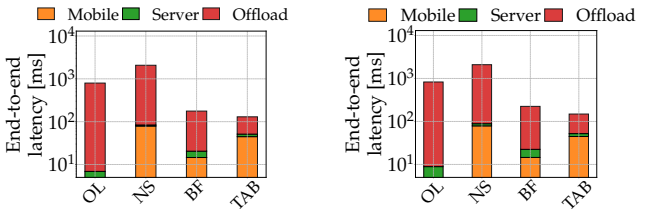
Offload data compression. Figure 8a and Figure 8b report the offloaded data size with different compression methods with ResNet-50 backbone in 'Office' experiment setup. Multi-TAB only transmits 7.8 KB of data in closer position 'A'. In farther position 'B', Multi-TAB transmits only 13.45 KB of data. This means the transformed input tensor of the encoder compresses up to 98.2% data in position 'A' and 97.84% compression in position 'B'. Compared to BF, Multi-TAB compresses 93.84% of the input data. Among all the baselines, Multi-TAB compresses the intermediate representation the most to sustain the processing latency. Compared to other baselines, BF keeps the offloaded data size constant due to its fixed bottleneck design. Each baseline model was trained to achieve accuracy $> 85\%$; specifically, Multi-TAB achieves 86.23% MODA, satisfying the $> 85.0\%$ MODA threshold.

Head models. Table I compares different head models along with two widely used CNN-based backbones specifically optimized for mobile devices, MobileNet [30] and EfficientNet [31]. The lightweight head architecture of Multi-TAB with stage-1 split point encoder contains only 0.210 million parameters, representing a 98.1% reduction compared to ResNet-18-stage4 11.18 million parameters. In addition, the significantly computational complexity of Multi-TAB translates directly into faster execution times on resource-constrained hardware, requiring only 17.9 ms per inference on a Jetson Nano and 157.0 ms on a Raspberry Pi 4, outperforming both MobileNet (33.01 ms and 281.24 ms, respectively) and EfficientNet (59.10 ms and 221.31 ms, respectively). These results clearly illustrate the superior practicality of Multi-TAB for deployment at the mobile device with tight resource budget.

Comparison with Task-oriented Feature Extraction. A detailed comparison with PIB [15] under varying communication bottlenecks is provided in Section IV; Multi-TAB consistently outperforms PIB at tight constraints (10 KB and 25 KB) by up to 2.3% MODA on WILDTRACK and up to 2.7% on MULTIVIEWX.

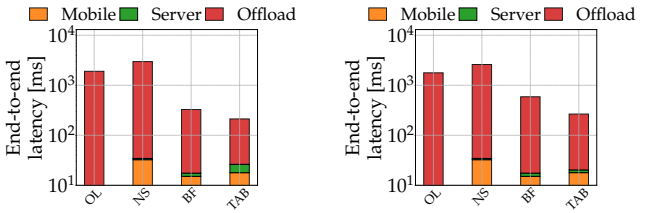
Head Models	FLOPs GMAC	Latency on JTX [ms]	Avg. Latency on RP4 [ms]
ResNet18-stage4	1.82	35.19 ±1.72	263.91 ±13.61
ResNet18-stage1	0.0718	17.89 ±0.24	157.07 ±45.12
MobileNetv2(full)	0.303	33.01 ±0.21	281.24 ±23.41
EfficientNet(full)	0.395	59.10 ±1.49	221.31 ±12.45

Table I: Comparison Between ResNet18-stage4, ResNet18-stage1, MobilenetV2, EfficientNet as head model deployment. JTX = Jetson Nano, RP4 = Raspberry Pi 4.



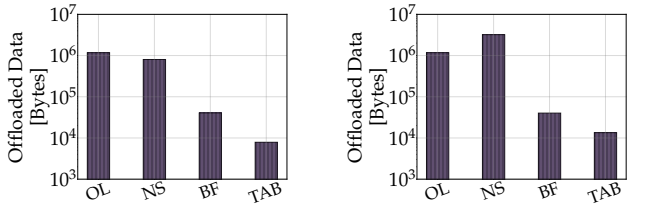
(a) '5 dB' Scenario - ResNet18 (b) '30 dB' Scenario - ResNet18

Figure 6: End-to-end task processing latency on Colosseum for '5 dB' and '30 dB' scenarios and ResNet18 as backbone. All the model task accuracy was $> 85.0\%$ MODA.



(a) Position 'A'- ResNet18 (b) Position 'B'- ResNet18

Figure 7: End-to-end task processing latency using Jetson Nano as mobile device and ResNet-18 as backbone. All the model task accuracy was $> 85.0\%$ MODA.



(a) Position 'A' - ResNet50 (b) Position 'B' - ResNet50

Figure 8: Evaluation of offloaded data size in office setup. We recorded the average offloaded data sizes for test images in the WILDTRACK dataset.

E. Ablation study

Loss components performance tradeoff. Table II reports the effect of the information bottleneck weight β and the masking-policy efficiency weight λ on MODA. We perform this experiment on our simulation testbed with a fixed SNR of 20 dB during testing and sampling from a range of 0 dB to 20 dB.**When** $\lambda = 0$**.** IB alone regularizes task-oriented feature compression and its impact is dataset-specific. On WILDTRACK, MODA increases from 86.25% at $\beta = 0$ to 86.61% at $\beta = 10^{-4}$ and 88.2% at $\beta = 10^{-3}$. On MULTIVIEWX, the difference is small, from 84.11% to 85.25% and 85.10%. **When** $\beta = 0$**.** MODA becomes highly sensitive to λ. On WILDTRACK, MODA drops from 86.25% at $\lambda = 0$ to 56.14% at $\lambda = 10^{-1}$, 56.22% at $\lambda = 10^{-2}$, and 58.56% at $\lambda = 10^{-3}$. On MULTIVIEWX, MODA drops from 84.11% at $\lambda = 0$ to 58.15%, 55.27%, and 72.56% for $\lambda \in \{10^{-1}, 10^{-2}, 10^{-3}\}$. *These results indicate that without IB regularization, the encoder encodes diverse features and accuracy degrades once the efficiency objective is introduced.*

Datasets	λ	$\beta = 0$	$\beta = 10^{-4}$	$\beta = 10^{-3}$
		MODA %	**MODA %**	**MODA %**
WILDTRACK	0	86.25	86.61	88.2
	10^{-1}	56.14	70.67	71.56
	10^{-2}	56.22	71.78	75.73
	10^{-3}	58.56	81.76	81.83
MULTIVIEWX	0	84.11	85.25	85.10
	10^{-1}	58.15	68.65	77.55
	10^{-2}	55.27	69.78	73.27
	10^{-3}	72.56	72.76	78.33

Table II: Simulation on trade-off between loss function components and performance for λ and β.

Joint effect of λ and β. A non-zero β consistently improves robustness for $\lambda > 0$. On WILDTRACK, at $\lambda = 10^{-2}$ MODA increases from 56.22% at $\beta = 0$ to 71.78% at $\beta = 10^{-4}$ and 75.73% at $\beta = 10^{-3}$, and at $\lambda = 10^{-3}$ it increases from 58.56% to 81.76% and 81.83%. On MULTIVIEWX, at $\lambda = 10^{-2}$ MODA increases from 55.27% to 69.78% and 73.27%, and at $\lambda = 10^{-3}$ it increases from 72.56% to 72.76% and 78.33%. *Overall, IB regularization concentrates task-relevant information and makes the representation more resilient when training with efficiency objective. There is sweet spot of choosing these two hyper parameters.*

F. Comparison with Task-oriented Baseline

Figure 9 reports emulation-based experiments where we train a PIB model under a fixed data-rate constraint of 250 KB and subsequently test its performance across various tighter communication bottlenecks (10 KB, 25 KB, etc.), which in our simulation settings corresponds to approximately -4 dB. In contrast, `Multi-TAB` is trained using a masking policy conditioned on a uniform SNR profile from 0 dB to 20 dB. To test generalization, we trained `Multi-TAB` and PIB on WILDTRACK and evaluated on MULTIVIEWX. On WILDTRACK, `Multi-TAB` consistently outperforms PIB under strict communication constraints: at the lowest bottlenecks of 10 KB and 25 KB, `Multi-TAB` achieves 2.3% and 1.8% higher MODA, respectively. On MULTIVIEWX, `Multi-TAB` obtains MODA improvements of 1.3% and 2.7%. The communication bottleneck was simulated by setting the link rate to emulate channel capacity with a comparable SNR value from the wireless channel model; in the testbed, this was achieved by limiting the Wi-Fi interface capacity via Linux commands. The key observation is that *because of variable-SNR-conditioned training,* `Multi-TAB` *achieves higher task performance even when trained once on a single SNR distribution profile. Other task-oriented communication baselines fail to adapt uniformly to different channel conditions.* `Multi-TAB` also achieves better generalization to the challenging MULTIVIEWX dataset even when not trained on it.

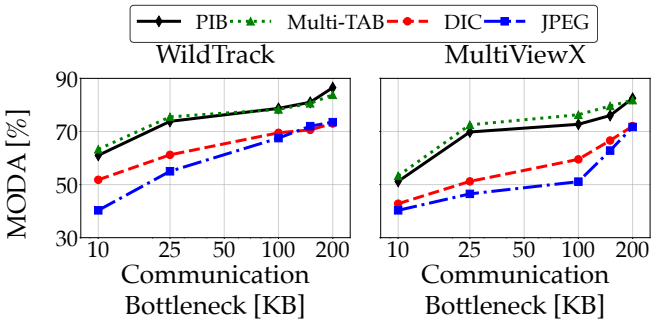


Figure 9: Evaluation of a simulation-based experiment under different communication bottlenecks using ResNet-18 backbone. The x-axis shows the communication bottleneck in KB. PIB is trained with a 250 KB bottleneck. JPEG quality was set to 50%.

V. Related Work

Feature Compression. Recent studies show that deep networks can jointly learn task-oriented feature extraction and compression. BottleNet [32] combines an on-device CNN with a JPEG encoder for image classification, while related methods address image retrieval [33], point-cloud analytics [34], and accuracy-preserving neural compression [35]. Although effective for reducing single-device offloading traffic, these methods ignore correlations among features from multiple devices, motivating distributed feature encoding.

The IB principle has been widely used to remove task-irrelevant redundancy in edge inference [21]. Prior works extend this idea to multi-device cooperative inference [36], edge video analytics with spatio-temporal redundancy [14], and priority-based route selection [15]. However, these frameworks do not directly support split inference because they depend on deep feature extractors, non-differentiable device-ranking routines, and fixed target-rate or SNR training profiles. In contrast, we propose a lightweight encoder that compresses task-relevant intermediate features and enables adaptive compression under dynamic wireless channels. **Split Computing.** Most existing approaches choose the split layer to balance edge–cloud computation [17]. Since convolutional activations and feature maps expand rapidly in early layers [37], practical systems favor split points at backbone bottlenecks. This has led to "bottleneck injection" [32, 38],

followed by retraining with only the task objective (e.g., cross-entropy) [39]. DeepOCD [40] further analyzes such splits and quantizes intermediate features for transmission. To compensate for capacity loss introduced by narrow latent spaces, later work augments training with Knowledge Distillation (KD) [41]. PhyDNN [42] is another IB-based framework that reduces transmission latency by executing DNN at the physical layer of the wireless network protocol. SplitNets [43], which statically partitions DNNs for single-device execution, unlike our framework enables SNR-aware, task-adaptive compression for multi-camera split computing under dynamic wireless conditions. All of this work addresses the communication vs. performance vs. resource consumption trade-off—but none tackles the multi-device edge inference problem. In our framework, we propose a distributed Variational Information Bottleneck (VIB)-based approach to efficiently tackle split computing in multi-device edge computing.

VI. Conclusion

In this paper, we presented `Multi-TAB`, a novel resource-aware split computing framework tailored for multi-view computer vision at the edge. By splitting pretrained DNNs into lightweight head models on mobile devices and robust tail models on servers, `Multi-TAB` effectively optimizes the trade-off between inference accuracy and communication overhead. A core innovation is the combination of an SNR-conditioned masking mechanism with an information bottleneck-based encoder-decoder, enabling adaptive compression of intermediate feature maps according to wireless channel conditions and view relevance. Through extensive experiments on pedestrian density detection using WILD-TRACK and MULTIVIEWX, `Multi-TAB` achieves up to **33.7%** lower end-to-end latency while maintaining accuracy within **2.4%**. Our implementation on the testbeds ensures the viability of `Multi-TAB`. Importantly, the masking mechanism ensures resilience under variable SNR profiles which is common in real-world deployments but seldom addressed by existing solutions. Overall, `Multi-TAB` advances collaborative inference by enabling latency-optimized multi-camera systems in dynamic, bandwidth-constrained environments. Codes will be available at https://github.com/tanzilhassan/Multi-TAB.git

Acknowledgment

This work has been supported by AFOSR under grant FA9550-23-1-0261; by ONR under grant N00014-23-1-2221; and by DARPA under Cooperative Agreement D25AC00374-00.

References

[1] X. Liu, W. Liu, T. Mei, and H. Ma, "A deep learning-based approach to progressive vehicle re-identification for urban surveillance," in *Proc. Eur. Conf. Comput. Vision (ECCV)*, (Amsterdam, Netherlands), Oct. 2016.

[2] D. Tao, Y. Guo, B. Yu, J. Pang, and Z. Yu, "Deep multi-view feature learning for person re-identification," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, pp. 2657–2666, July. 2017.

[3] Y. Hou, L. Zheng, and S. Gould, "Multiview detection with feature perspective transformation," in *ECCV*, pp. 1–18, 2020.

[4] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *Proc. Int. Conf. Comput. Vision*, (Santiago, Chile), Dec. 2015.

[5] Y. Xu, X. Liu, Y. Liu, and S.-C. Zhu, "Multi-view people tracking via hierarchical trajectory composition," in *Proc. Conf. Comput. Vision Pattern Recognit.*, (Las Vegas, NV, USA), Jun. 2016.

[6] M. Palena, T. Cerquitelli, and C. F. Chiasserini, "Edge-device collaborative computing for multi-view classification," *Computer Networks*, vol. 254, p. 110823, 2024.

[7] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese, "3d-r2n2: A unified approach for single and multi-view 3d object reconstruction," in *Proc. Eur. Conf. Comput. Vision (ECCV)*, (Amsterdam, Netherlands), Oct. 2016.

[8] J. D. N. Dionisio, W. G. B. III, and R. Gilbert, "3D Virtual Worlds and the Metaverse: Current Status and Future Possibilities," *ACM Computing Surveys (CSUR)*, vol. 45, no. 3, pp. 1–38, 2013.

[9] H. Liu, Y. Tian, Y. Yang, L. Pang, and T. Huang, "Deep relative distance learning: Tell the difference between similar vehicles," in *Proc. Conf. Computer Vision Pattern Recognit.*, (Las Vegas, NV, USA), pp. 2167–2175, Jun. 2016.

[10] Y. Liu, J. Yan, F. Jia, S. Li, Q. Gao, T. Wang, X. Zhang, and J. Sun, "Petrv2: A unified framework for 3d perception from multi-camera images," *arXiv preprint arXiv:2206.01256*, 2022.

[11] Y. Liu, T. Wang, X. Zhang, and J. Sun, "Petr: Position embedding transformation for multi-view 3d object detection," *arXiv preprint arXiv:2203.05625*, 2022.

[12] M. Ye, J. Shen, G. Lin, T. Xiang, L. Shao, and S. C. Hoi, "Deep learning for person re-identification: A survey and outlook," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 6, pp. 2872–2893, 2021.

[13] Y. Matsubara, M. Levorato, and F. Restuccia, "Split Computing and Early Exiting for Deep Learning Applications: Survey and Research Challenges," *ACM Computing Surveys (CSUR)*, 2021.

[14] J. Shao, X. Zhang, and J. Zhang, "Task-oriented communication for edge video analytics," *IEEE Transactions on Wireless Communications*, vol. 23, no. 5, pp. 4141–4154, May 2024.

[15] Z. Fang, S. Hu, J. Wang, Y. Deng, X. Chen, and Y. Fang, "Prioritized information bottleneck theoretic framework with distributed online learning for edge video analytics," *IEEE Transactions on Networking*, 2025.

[16] Y. Matsubara, D. Callegaro, S. Singh, M. Levorato, and F. Restuccia, "BottleFit: Learning Compressed Representations in Deep Neural Networks for Effective and Efficient Split Computing," in *Proceedings of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2022.

[17] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[18] G. K. Wallace, "The JPEG still picture compression standard," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii–xxxiv, Feb. 1992.

[19] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, "Variational image compression with a scale hyperprior," in *Proc. Int. Conf. Learn. Repr.*, 2018.

[20] L. Bonati, P. Johari, M. Polese, S. D'Oro, S. Mohanti, M. Tehrani-Moayyed, D. Villa, S. Shrivastava, C. Tassie, K. Yoder, A. Bagga, P. Patel, V. Petkov, M. Seltser, F. Restuccia,

A. Gosain, K. R. Chowdhury, S. Basagni, and T. Melodia, "Colosseum: Large-Scale Wireless Experimentation Through Hardware-in-the-Loop Network Emulation," in *2021 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pp. 105–113, IEEE, 2021.

[21] N. Tishby, F. C. Pereira, and W. Bialek, "The information bottleneck method," in *Proc. Annu. Allerton Conf. Commun. Control Comput.*, (Monticello, IL, USA), Oct. 2000.

[22] I. E. Aguerri and A. Zaidi, "Distributed variational representation learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, pp. 120–138, Jan. 2021.

[23] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *J. Amer. Statist. Assoc.*, vol. 112, pp. 859–877, Jul. 2017.

[24] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *Proc. Int. Conf. Learn. Repr. (ICLR)*, (Banff, Canada), Apr. 2014.

[25] W. Kool, H. van Hoof, and M. Welling, "Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement," in *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019.

[26] Y. Hou, L. Zheng, and S. Gould, "Multiview detection with feature perspective transformation," in *Eur. Conf. Comput. Vision*, Aug. 2020. [Online]. Available: https://www.ecva.net/papers/eccv_2020/papers_ECCV/papers/123520001.pdf.

[27] R. Kasturi, D. Goldgof, P. Soundararajan, V. Manohar, J. Garofolo, R. Bowers, M. Boonstra, V. Korzhova, and J. Zhang, "Framework for performance evaluation of face, text, and vehicle detection and tracking in video: Data, metrics, and protocol," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, pp. 319–336, Mar. 2008.

[28] D. Villa, M. Tehrani-Moayyed, C. P. Robinson, L. Bonati, P. Johari, M. Polese, and T. Melodia, "Colosseum as a digital twin: Bridging real-world experimentation and wireless network emulation," *IEEE Transactions on Mobile Computing*, vol. 23, no. 10, pp. 9150–9166, 2024.

[29] J. Bégaint, F. Racapé, S. Feltman, and A. Pushparaja, "CompressAI: A pytorch library and evaluation platform for end-to-end compression research," *arXiv preprint arXiv:2011.03029*, 2020. [Online]. Available: https://arxiv.org/abs/2011.03029.

[30] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *IEEE CVPR*, pp. 4510–4520, 2018.

[31] B. Koonce, "Efficientnet," in *Convolutional neural networks with swift for Tensorflow: image recognition and dataset categorization*, pp. 109–123, Springer, 2021.

[32] A. E. Eshratifar, A. Esmaili, and M. Pedram, "Bottlenet: A deep learning architecture for intelligent mobile cloud computing services," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, pp. 1–6, 2019.

[33] M. Jankowski, D. Gündüz, and K. Mikolajczyk, "Wireless image retrieval at the edge," *IEEE J. Sel. Areas Commun.*, vol. 39, pp. 89–100, May 2021.

[34] J. Shao, H. Zhang, Y. Mao, and J. Zhang, "Branchy-gnn: A device-edge co-inference framework for efficient point cloud processing," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, (Toronto, Canada), Jun. 2021.

[35] Y. Dubois, B. Bloem-Reddy, K. Ullrich, and C. J. Maddison, "Lossy compression for lossless prediction," in *Proc. Int. Conf. Learn. Repr. Workshop on Neural Compression*, (Vienna, Austria), May 2021.

[36] J. Shao, Y. Mao, and J. Zhang, "Task-oriented communication for multidevice cooperative edge inference," *IEEE Transactions on Wireless Communications*, vol. 22, no. 1, pp. 73–87, 2022.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[38] J. Shao and J. Zhang, "Bottlenet++: An end-to-end approach for feature compression in device-edge co-inference systems," in *Proc. IEEE Int. Conf. Commun. Workshop*, (Dublin, Ireland), Jun. 2020.

[39] Y. Matsubara, M. Levorato, and F. Restuccia, "Split computing and early exiting for deep learning applications: Survey and research challenges," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–30, 2022.

[40] J. Emmons, S. Fouladi, G. Ananthanarayanan, S. Venkataraman, S. Savarese, and K. Winstein, "Cracking open the dnn black-box: Video analytics with dnns across the camera-cloud boundary," in *Proceedings of the 2019 workshop on hot topics in video analytics and intelligent edges*, pp. 27–32, 2019.

[41] Y. Matsubara, S. Baidya, D. Callegaro, M. Levorato, and S. Singh, "Distilled split deep neural networks for edge-assisted real-time systems," in *Proc. the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges*, pp. 21–26, ACM, 2019.

[42] M. Abdi, K. F. Haque, F. Meneghello, J. Ashdown, and F. Restuccia, "Phydnns: Bringing deep neural networks to the physical layer," in *IEEE INFOCOM 2025-IEEE Conference on Computer Communications*, pp. 1–10, IEEE, 2025.

[43] X. Dong, B. De Salvo, M. Li, C. Liu, Z. Qu, H.-T. Kung, and Z. Li, "Splitnets: Designing neural architectures for efficient distributed computing on head-mounted systems," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12559–12569, 2022.