Why Should the Server Do It All?: A Scalable, Versatile, and Model-Agnostic Framework for Server-Light DNN Inference over Massively Distributed Clients via Training-Free Intermediate Feature Compression

Mingyu Sung, Suhwan Im, Daeho Bang, Il-Min Kim, Sangseok Yun, and Jae-Mo Kang[‡]

Abstract

Modern DNNs often rely on edge–cloud model partitioning (MP), but widely used schemes fix shallow, static split points that underutilize edge compute and concentrate latency and energy on the server. The problem is exacerbated in autoregressive (AR) LLM inference, where per-token forward passes repeatedly generate bulky intermediate features (IFs). We introduce SLICER, a retraining-free, architecture-agnostic framework that compresses IFs to reduce both communication and server load in split computing. SLICER combines (i) asymmetric top-K filtering (ATKF) to sparsify low-magnitude activations, (ii) magnitude-splitting (MS) to group the remaining non-zeros into equal-cardinality blocks, and (iii) adaptive bit quantization (ABQ) that selects per-block bitwidths under a distortion budget. Across standard vision and LLM workloads (e.g., ImageNet/COCO; HellaSwag, PIQA, ARC-E/C, GSM8K, HumanEval), SLICER reduces uplink volume by up to 10× and server GPU time by up to 4.4×, while keeping task quality within 0–3 pp of baseline. In multi-device settings and AR LLMs, SLICER scales by shifting meaningful compute to the edge and lowering bits-per-token and server time per token, stabilizing per-step traffic. The codec attaches to off-the-shelf models without retraining or architectural changes, offering a plug-and-play path to scalable, low-latency distributed inference. Code is provided in the supplementary material.

1 Introduction

Deep neural network (DNN) has advanced at an unprecedented pace, primarily enabled by the ability to train deeper neural networks. This breakthrough was largely facilitated by *residual learning* [1], where skip connections mitigate the vanishing gradient problem and allow the convergence of networks with hundreds or even thousands of layers. Building on this foundation, both the depth and width of neural networks have been scaled, achieving state-of-the-art (SOTA) performance [2, 3, 4, 5, 6, 7, 8]. However, deploying these high-performance models across diverse hardware platforms remains a significant challenge, as their computational and memory demands often surpass the capabilities of target devices.

1.1 Related Works and Motivations

[*]Mingyu Sung, Suhwan Im, Daeho Bang and Jae-Mo Kang are with the Department of Artificial Intelligence, Kyungpook National University, Daegu, South Korea (Corresponding author: Jae-Mo Kang, e-mail: jmkang@knu.ac.kr).

[†]Il-Min Kim is with the Department of Electrical and Computer Engineering, Queen's University, Kingston, K7L 3N6, Canada.

[‡]Sangseok Yun is with the Department of Information and Communications Engineering, Pukyong National University, Busan 48513, South Korea (Corresponding author: Sangseok Yun, e-mail: ssyun@pknu.ac.kr).

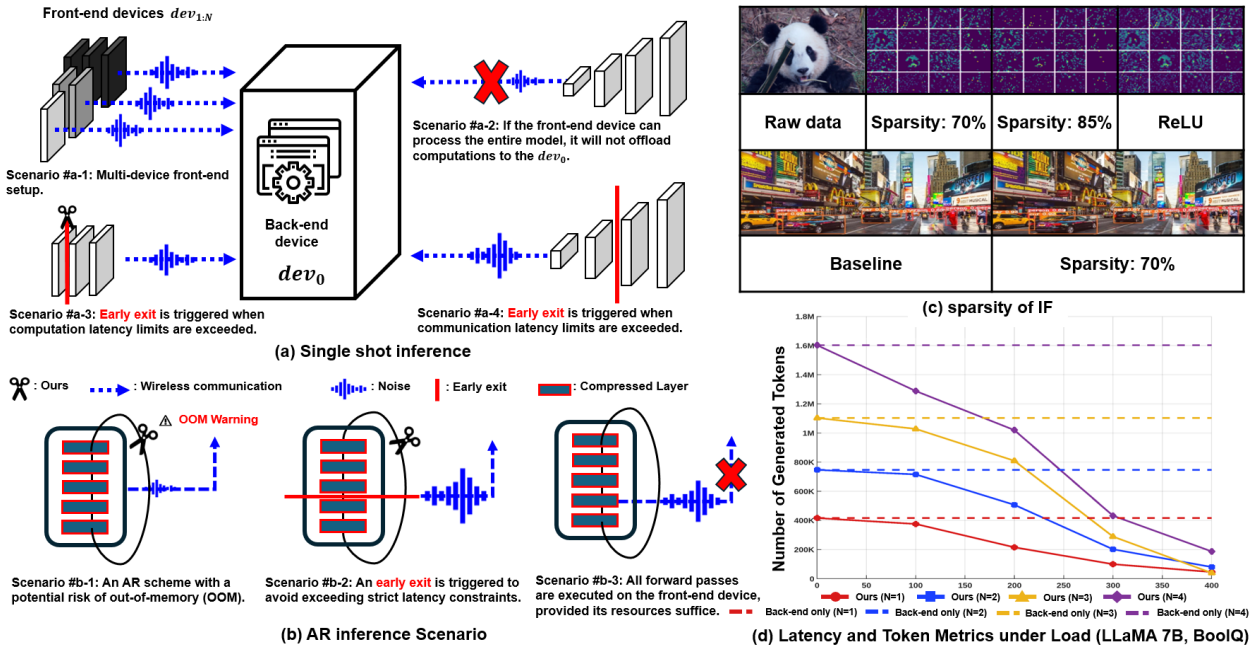

Figure 1: (a) Single-shot inference model where multiple front-end devices offload tasks to a shared back-end based on latency constraints. (b) AR inference scenarios illustrating early exits triggered by memory or latency limits. (c) Visualization of sparsity in IFs, a key technique for compression. (d) Performance metrics demonstrating that our approach reduces the back-end server's load as the number of front-end devices increases.

DNN Model Partitioning (MP). Within edge computing (EC), a common baseline is to transmit raw inputs to a back-end server that executes the entire DNN [9]. While this preserves model accuracy and offloads device compute, it can cause (1) server congestion as the number of clients grows [10], (2) higher end-to-end (E2E) latency under adverse networks [11], and (3) underutilization of increasingly capable edge accelerators [12]. Consequently, efficient distributed inference techniques such as model partitioning (MP) are essential to bridge the resource gap for deploying neural models on edge devices. To mitigate these issues, *split computing* (SC)—a widely studied *subclass* of MP—partitions the network into: (i) early layers executed on the front-end (edge) and (ii) the remaining layers on the back-end (server). Instead of raw inputs, the front-end transmits *intermediate features* (IFs), which can reduce bandwidth and E2E latency. With an appropriate split point, SC may also reduce the exposure of raw content in transit [9].

Cloud-Centric Bias in MP and Its Amplification in LLMs. A critical flaw in many existing MP policies is their reactive, cloud-centric nature. In response to deteriorating wireless links, they often default to shifting the split point closer to the input [11, 13], a strategy that offloads the vast majority of the computational burden to the server. While this may reduce communication latency for a single client, it becomes an unscalable, server-heavy approach in realistic multi-device deployments, amplifying queueing delays and operational costs [12, 14].

This problem is severely magnified in modern LLM services. Here, naive, server-centric offloading underutilizes powerful edge devices and quickly overloads shared servers [15, 16], an effect we quantify in Fig. 1(d). The challenge is further exacerbated by the nature of autoregressive (AR) inference, which introduces two unique problems: (1) per-token full-stack execution diminishes the amortized benefits of shallow splits, and (2) accumulating traffic and state mean the total data volume grows with the output length.

Unfortunately, prior MP work on IF compression is almost entirely vision-centric, focusing on single-shot payload reduction [17, 18, 19, 20, 21]. These methods are ill-suited for the token-by-token nature of AR workloads, where continuous, per-step bandwidth savings are crucial for maintaining low latency. The limitations of existing cloud-centric policies and the unsuitability of vision-focused codecs for AR workloads highlight a clear need for a new approach.

1.2 Contributions

To address this gap, in this work, we introduce *SLICER*, a training-free, task-agnostic framework for split computing that couples a lightweight IF codec with a predictive, constraint-aware configuration. Our key technical contributions and breakthroughs in this work include the followings:

- **Lightweight IF Compression & Balanced Utilization.** We propose a lightweight IF compression scheme that effectively exploits sparsity, suppresses quantization error, and optimizes bit-width. Implemented with fused CUDA kernels for low overhead, its compression ratio is precisely tunable to match diverse model partitioning (MP) scenarios.
- **Generality Across Vision & NLP Tasks.** Our framework is designed to be task-agnostic. A single codec handles both vision and NLP workloads, which eliminates redundant, specialized components and simplifies deployment across heterogeneous devices.
- **Plug-and-Play Integration.** Our codec is plug-and-play, attaching to off-the-shelf models without retraining or architectural changes. Validated across extensive vision and LLM benchmarks, it reduces server-side latency by up to 4.4× and shrinks bandwidth substantially while preserving model accuracy—all without requiring model modification.

2 Our Approach: SLICER

2.1 Partition Layer Selection Strategy for Server-Light DNN Inference

DNN Inference Latency. Consider a DNN $\mathbb{L} = \{\mathbb{L}_1, \mathbb{L}_2, \ldots, \mathbb{L}_{\hat{\ell}}\}$ consisting of $\hat{\ell}$ layers, where $\mathbb{L}_\ell$ denotes the ℓ-th layer. Define the partial network $\mathbb{L}_{1:\ell}$ as the sequential composition of layers $\mathbb{L}_1, \ldots, \mathbb{L}_\ell$. The IF is given by

$$\mathbf{x}_\ell \;=\; \mathbb{L}_{1:\ell}(\mathbf{x}).$$

Given a selected split point ℓ, the IF $\mathbf{x}_\ell$ is transmitted to the back-end device (dev_0). Each front-end device communicates over a noisy wireless channel whose quality may vary from node to node. Thus, we define the communication latency using the ε-outage latency [22]:

$$L_c(\ell, R) = \frac{B(\mathbf{x}_\ell)}{R}\left\lceil \frac{\ln \varepsilon}{\ln P_o(R)} \right\rceil + \zeta(\mathbf{x}_\ell), \qquad P_o(R) = 1 - \exp\Big(-\tfrac{2^{R/W}-1}{\gamma}\Big). \tag{1}$$

where $B(\mathbf{x}_\ell)$ is the bit-length of the IF, R denotes the transmission rate, W the channel bandwidth, and γ the average signal-to-noise ratio (SNR). Additionally, $\zeta(\mathbf{x}_\ell)$ represents the time required to encode the IF $\mathbf{x}_\ell$ prior to transmission. We define the computation latency $L_d(\mathbb{L}_{1:\ell})$ as the processing time incurred by the front-end device[4]. Hence, the total latency is given by:

$$L(\ell, R) = L_d\big(\mathbb{L}_{1:\ell}\big) + L_c(\ell, R). \tag{2}$$

DNN Partitioning Approach with Multiple Devices. We consider a deployment consisting of one back-end device and N front-end devices $\{\text{dev}_0; \text{dev}_1, \ldots, \text{dev}_N\}$.

- **Back-end device (dev_0):** Executes all layers beyond the split point and completes the inference.
- **Front-end devices ($\text{dev}_{1:N}$):** Each front-end device processes layers up to the chosen split point and transmits the IF to dev_0 for further processing.

Single-Shot DNN Inference Scenario. *Scenario #a-1* in Fig. 1(a) illustrates the multi-device scenario. A front-end device, dev_i, with memory budget $\mathcal{M}$, aims to execute the maximum number of layers while satisfying the latency constraint $\mathcal{D}$. For a DNN of $\hat{\ell}$ layers, let $m(\mathbb{L}_{1:\ell})$ denote the cumulative memory requirement of layers 1:ℓ. We select the deepest feasible split point ℓ^* that satisfies both latency and memory constraints:

$$\ell^* = \max\Big\{\ell \in \{1, \ldots, \hat{\ell}\} \;\Big|\; m\big(\mathbb{L}_{1:\ell}\big) \le \mathcal{M}, L(\ell, R) \le \mathcal{D}\Big\}. \tag{3}$$

The front-end device executes layers $\mathbb{L}_{1:\ell^*}$ and transmits the IF to dev_0, which completes inference by executing layers $\mathbb{L}_{\ell^*+1:\hat{\ell}}$. If the wireless link degrades such that $L(\ell^*, R) > \mathcal{D}$, Eq. 3 automatically selects a shallower split point, thereby satisfying both latency and memory constraints.

[4]The back-end device bypasses complex delay modeling by assigning each front-end a delay constraint that incorporates the back-end's own status.

Autoregressive DNN Inference Scenario. AR inference iteratively feeds the model's outputs back into its inputs, requiring repeated forward passes. Under these conditions, the partition strategy described in Eq. 3 may become impractical as memory consumption increases with each decoding step. To manage this memory growth, we adopt a compressed network $\lessdot_{1:\hat{\ell}}$. Let $C(\mathbb{L}_{1:\hat{\ell}}; P)$ denote the overall performance after applying the compression parameter P to all layers. Accounting for the extra memory required for repeated forward passes, $\mathcal{M}_{ar}$, we aim to find the parameter $P_{\max}$ that maximizes performance:

$$P_{\max} = \arg\max_{P} C\big(\mathbb{L}_{1:\hat{\ell}}; P\big) \quad \text{s.t.} \quad m\big(\lessdot_{1:\hat{\ell}}\big) + \mathcal{M}_{ar} \leq \mathcal{M}. \tag{4}$$

Satisfying this constraint keeps the compressed network memory-feasible throughout AR decoding, preventing out-of-memory (OOM) errors while maintaining high performance.

For the first $(w-1)$ steps, the dev_i processes all compressed layers locally (i.e., $\lessdot_{1:\hat{\ell}}$). This incurs a per-step cost of $L_d\big(\lessdot_{1:\hat{\ell}}\big)$ on dev_i. On the final (w-th) step, however, the dev_i executes only up to ℓ, then offloads the IF $\mathbf{x}_\ell^w$ to dev_0. Consequently, the total latency is

$$L_{\text{AR}}(\ell, w, R) = \underbrace{(w-1)\, L_d\big(\lessdot_{1:\hat{\ell}}\big)}_{\text{full passes}} + \underbrace{L_d\big(\lessdot_{1:\ell}\big)}_{\text{partial pass}} + \underbrace{L_c\big(\ell, R\big)}_{\text{offloading IF}}. \tag{5}$$

Eq. 5 represents the total latency for a split point (ℓ, w). By requiring this latency to remain below the budget $\mathcal{D}$ and the offloaded feature size to stay within the per-step memory cap $\mathcal{M}_{\text{ar}}$, we then select the deepest admissible split point (w^*, ℓ^*) as follows:

$$(w^*, \ell^*) = \max_{w,\,\ell} \Big\{ (w, \ell) \;\Big|\; L_{\text{AR}}(\ell, w, R) \leq \mathcal{D},\; B\big(\mathbf{x}_\ell^w\big) \leq \mathcal{M}_{\text{ar}} \Big\}, \tag{6}$$

which consistently meets both the latency and the memory constraints while maximizing computation on the front-end devices, $dev_{i:N}$. The strategy in Eq. 6 works as follows. When the memory or latency budget becomes tighter, each dev_i moves to the next shallower split point *Scenario #b-1, #b-2* in Fig. 1(a), maximizing computation on the device. In contrast, when resources are sufficient, there is no need to offload to dev_0 *Scenario #b-3*, reducing round-trip traffic and avoiding unnecessary queueing delays. In this way, the system adapts on the fly to channel quality and device heterogeneity, balancing edge computation with minimal server assistance throughout AR inference.

3 Further Breakthrough: Intermediate Feature Compression in SLICER

Shrinking the IF is essential for practical model partitioning, especially in today's dynamic environments running mixed vision and NLP workloads. An effective compression scheme for this context must meet four strict criteria that conventional methods fail to satisfy simultaneously: (1) guaranteeing an exact compression ratio, (2) controlling this ratio at runtime, (3) operating at real-time speed, and (4) remaining task-agnostic.

Existing techniques fall short, as they were originally developed for a different purpose: static model weight compression for inference optimization, not dynamic IF compression. This fundamental design mismatch makes them unable to accurately capture a target compression ratio, a strict requirement for dynamic MP but a loose one for offline weight pruning. Moreover, since they are designed for a one-time, offline application, they inherently lack the mechanisms to dynamically control the compression level at runtime. Finally, many of these approaches are tailored to specific data modalities (e.g. vision), failing the task-agnostic requirement.

To meet all four demands, we propose a novel, unified, and training-free pipeline. Our asymmetric top-K filtering (ATKF) provides deterministic control for an exact compression target, while our adaptive bit quantization (ABQ) uses lightweight integer operations for real-time, per-block bit selection. Crucially, our entire task-agnostic framework, combined with magnitude-splitting (MS), delivers the precise, fast, and universally applicable control required for modern MP workloads. A comprehensive comparison highlighting these distinctions is provided in Appendix **??**. In addition, a detailed example of each stage can be found in the appendix **??**.

3.1 Asymmetric Top-K Filtering (ATKF)

Let $x \in \mathbb{R}^{H \times W}$ with $N = HW$ elements and target sparsity $s \in [0, 1]$ and asymmetry factor $\lambda \in [0, 1)$. Define

$$k_{\text{keep}} = \big\lfloor (1-s)\, N \big\rfloor, \qquad \tau = k_{\text{keep}}\text{-th largest } |x_{i,j}|, \qquad \tau_+ = (1+\lambda)\,\tau, \qquad \tau_- = -(1-\lambda)\,\tau.$$

Strictly retained indices are

$$S_{\text{strict}} = \{(i,j) \mid x_{i,j} > \tau_+ \text{ or } x_{i,j} < \tau_-\}.$$

If $|S_{\text{strict}}| < k_{\text{keep}}$, randomly select additional indices T from $\{(i,j) \mid |x_{i,j}| = \tau\}$ until the total number of retained elements equals k_{keep}, i.e., $|S_{\text{strict}} \cup T| = k_{\text{keep}}$.

$$\mathcal{F}_{i,j}(x,s,\lambda) = \begin{cases} x_{i,j}, & (i,j) \in S_{\text{strict}} \cup T, \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

Eq. 7 keeps the k_{keep} largest-magnitude coefficients and sets the remaining entries to zero, thereby achieving the exact nonzero count k_{keep} (i.e., a nonzero fraction $1-s$) while preserving shape. Note that if ties at $|x| = \tau$ are insufficient, we continue filling from the next lower magnitudes in descending $|x|$ (breaking ties at random) until $|S_{\text{strict}} \cup T| = k_{\text{keep}}$. The asymmetry factor λ shifts the positive and negative cut-offs. Because only low-magnitude coefficients are suppressed—and ties at $|x| = \tau$ are resolved by random sampling—the split filter achieves the target sparsity with minimal impact on model accuracy and communication cost.

3.2 Magnitude-Splitting (MS) with Equal Cardinality

To further reduce communication costs while preserving structural information, we apply a magnitude-based partitioning of the non-zero entries. Unlike typical thresholding schemes that define uniform intervals in the value domain, our approach ensures that each sub-tensor receives *same number of non-zero entries* in sorted order.

Pre-Step: Sign Separation. Given an IF $X \in \mathbb{R}^{N\times K}$ after *ATKF*, define

$$X^{(+)} = \max(X, 0), \quad X^{(-)} = \max(-X, 0), \quad X = X^{(+)} - X^{(-)}.$$

The remainder of this section is executed twice—once for $X^{(+)}$ and once for $X^{(-)}$-with identical logic.

Step 1. Flatten, Collect non-zeros, Sort by magnitude. Let $X^{(s)}$ be either sign ($s \in \{+,-\}$) and $X^{(s)}_{\text{flat}} \in \mathbb{R}^T$ its flattened view ($T = NK$) [5]. Non-zero indices $\Omega^{(s)} = \{\, i \mid X^{(s)}_{\text{flat}}(i) \neq 0\}$ are extracted and sorted in descending absolute value:

$$(X^{(s)}_{\text{nz}}, \boldsymbol{\pi}^{(s)}) = \text{Sort}\big(|X^{(s)}_{\text{flat}}(i)| \mid i \in \Omega^{(s)}\big) \quad (\textit{sorted descendingly}).$$

Step 2. Equal-Cardinality partitioning. For a target block count $M^{(s)}$, set $\mathcal{K}^{(s)} = \lfloor |\Omega^{(s)}|/M^{(s)} \rfloor$. Then, we can define the index ranges $I^{(s)}_m$ for $m = 1, \ldots, M^{(s)}$. Each block is re-inserted into a zero-initialized tensor $\mathbf{y}$ at positions indicated by $\boldsymbol{\pi}^{(s)}(i)$.

Step 3. Compressed Sparse Row (CSR) Encoding. Every block $X^{(s)}_m$ is stored using the CSR format: $\text{CSR}(X^{(s)}_m) = (\mathbf{v}^{(s)}_m, \mathbf{c}^{(s)}_m, \mathbf{r}^{(s)}_m)$.

3.2.1 Adaptive Bit Quantization (ABQ)

Asymmetric Integer Quantization (AIQ). After MS, the non-zero values $\mathbf{v}_m$ in each block are quantized independently using Asymmetric Integer Quantization (AIQ) with a computed zero-point and q_m bits:

$$\widehat{\mathbf{v}}_m = \left\lceil \tfrac{\mathbf{v}_m}{o_m} + \mathbf{z}_m \right\rfloor, \qquad o_m = \frac{v^{\max}_m - v^{\min}_m}{2^{q_m} - 1}, \qquad \mathbf{z}_m = \left\lceil \tfrac{v^{\min}_m}{o_m} \right\rfloor,$$

where $v^{\max}_m = \max(\mathbf{v}_m)$ and $v^{\min}_m = \min(\mathbf{v}_m)$. Since MS has already pruned extreme outliers, this per-block AIQ step introduces only minor distortion. The server reconstructs the tensor $\tilde{X}$ by reversing this process:

$$\tilde{X} \;=\; \Sigma\,\text{csr}^{-1}\big(\tilde{\mathbf{v}}_m, \mathbf{c}_m, \mathbf{r}_m\big), \quad \tilde{\mathbf{v}}_m \;=\; (\hat{\mathbf{v}}_m - \mathbf{z}_m) \cdot o_m. \tag{8}$$

[5] For a general IF $X \in \mathbb{R}^{N\times K}$, ATKF is applied to the flattened tensor of length $T = NK$; thus $k_{\text{keep}} = \lfloor (1-s)T \rfloor$.

ABQ as a Rate-Distortion Solver. To determine the optimal bit-width q_m for each block, ABQ uses a lightweight "integer mismatch" metric, $DS(\cdot)$, instead of MSE (see Appendix **??**):

$$DS(T_1, q_1, T_2, q_2) \;=\; \frac{1}{n}\sum \left| \left\lfloor \frac{T_1}{2^{(q_1-q_2)}} \right\rfloor - T_2 \right|. \tag{9}$$

The ABQ algorithm greedily lowers the bit precision q from an initial maximum q_{bit} until a user-defined distortion threshold δ is met. Formally, we solve:

$$\min_{q \le q_{\text{bit}}} \quad q \quad \text{s.t.} \quad DS\big(\widehat{\mathbf{T}}_0, q_{\text{bit}}, \widehat{\mathbf{T}}, q\big) \;\le\; \delta,$$

where $\widehat{\mathbf{T}}_0$ is the reference integer tensor. Because this search operates entirely on integers, it is computationally lightweight, allowing ABQ to efficiently balance bit usage and distortion at runtime.

3.3 Constraint-Aware Predictive Configuration (SLICER-Search)

Given real-time budgets $(\mathcal{D}, \mathcal{M}, \mathcal{M}_{\text{ar}}, \mathcal{M}_{\text{buf}}, R)$ and a candidate split (single-shot: ℓ; AR: (ℓ, w)), we choose compression knobs $\theta = \{s,\, M^{(+)}, M^{(-)},\, \lambda\}$ to (i) minimize accuracy loss while (ii) strictly satisfying latency and memory/throughput constraints. ABQ per-block bitwidths $\mathbf{q}^{(s)} = \{q_m^{(s)}\}$ are determined by an integer-only distortion guard δ (Sec. 3.2.1).

Predictive Bit/Time Model. Let the IF after ATKF/MS have $T = NK$ elements in total. ATKF fixes the nonzero count as $k_{\text{keep}} = \lfloor (1-s)T \rfloor$. MS then partitions nonzeros of each sign into $M^{(s)}$ equal-cardinality blocks, so each block has a known cardinality $|\Omega_m^{(s)}|$. Because ABQ selects $q_m^{(s)}$ adaptively from data, we predict the payload by a conservative upper bound that sets $q_m^{(s)} = q_{\text{bit}}$:

$$\widehat{B}^{\text{UB}}(\mathbf{x}_\ell; \theta) = \underbrace{\sum_{s\in\{+,-\}} \sum_{m=1}^{M^{(s)}} |\Omega_m^{(s)}|\, q_{\text{bit}}}_{\text{value bits}} \;+\; \underbrace{B_{\text{idx}}(N, K, M^{(+)}, M^{(-)}) + B_{\text{meta}}(M^{(+)}, M^{(-)})}_{\text{CSR/header}}.$$

Accordingly, we redefine the communication latency from Eq. equation 1 as:

$$\widehat{L}_c(\ell, R; \theta, \delta) = \frac{\widehat{B}^{\text{UB}}(\mathbf{x}_\ell; \theta)}{R} \left\lceil \frac{\ln \varepsilon}{\ln P_o(R)} \right\rceil + \widehat{\zeta}(\theta, \delta),$$

where $\widehat{\zeta}(\theta, \delta) = t_{\text{ATKF}}(s, \lambda) + t_{\text{MS}}(M^{(+)}, M^{(-)}) + \sum_{s,m} t_{\text{ABQ}}(q_m^{(s)})$ is a lightweight integer-operation time model, estimated *pre-execution* from device-specific lookup tables (calibrated offline).

Constraints. We adopt $\mathcal{M}_{\text{ar}}$ as the per-step offload-buffer cap (same unit as bits; consistent with Eq. 6).

$$\begin{aligned} &\text{single-shot:} \quad L(\ell, R; \theta, \delta) = L_d(\mathbb{L}_{1:\ell}) + \widehat{L}_c(\ell, R; \theta, \delta) \;\le\; \mathcal{D}, \quad m(\mathbb{L}_{1:\ell}) + m_{\text{buf}}(\theta, \delta) \;\le\; \mathcal{M}, \\ &\text{AR:} \qquad\qquad L_{\text{AR}}(\ell, w, R; \theta, \delta) \;\le\; \mathcal{D}, \quad \widehat{B}^{\text{UB}}(\mathbf{x}_\ell^w; \theta) \;\le\; \mathcal{M}_{\text{ar}}. \end{aligned}$$

Hierarchical Search Policy (Low-to-High Impact on Accuracy). (1) Tune $M^{(+)}$, $M^{(-)}$ (granularity $\leftrightarrow$ header trade-off) $\rightarrow$ (2) Increase s only if needed $\rightarrow$ (3) Retreat split ($\ell\downarrow$ or $w\downarrow$) and restart (1)–(2). This yields predictive upper-bound $\widehat{B}^{\text{UB}}$ and $\widehat{L}_c$ for pre-execution feasibility without trial-and-error.

4 Experiments

4.1 Experimental setup

We evaluate our framework on a wide range of datasets, tasks, and backbones. The vision task includes MobileNet-V2 [23], ResNet [1], CaiT [3], LSTR [24], and YOLOv6s [25] trained on CIFAR-100 [26], ImageNet-2012 [27], and COCO [28]. For NLP, we use Llama 2 [29], RoBERTa [30], QWen2.5 [31] and Deepseek [32] evaluated on GLUE [33], HellaSwag (HS) [34], PIQA [35], ARC-E/C [36], GSM8K [37] and HumanEval[38]. Action recognition is evaluated on LSTR on THUMOS'14 [39]. Unless otherwise specified, we set $\varepsilon = 0.001$, $W = 10\,\text{MHz}$, $\sigma_h^2 = 1$, $\gamma = 10$, $\delta = 0.01$, and $\lambda = 0$. All reported latencies are dataset-averaged values in milliseconds. We employ multi-level quantization: $Q = [q_1, \ldots, q_M]$ splits each IF into M sub-tensors and quantizes them with $q_1 > \ldots > q_M$ bits assigned in descending order of activation magnitude.

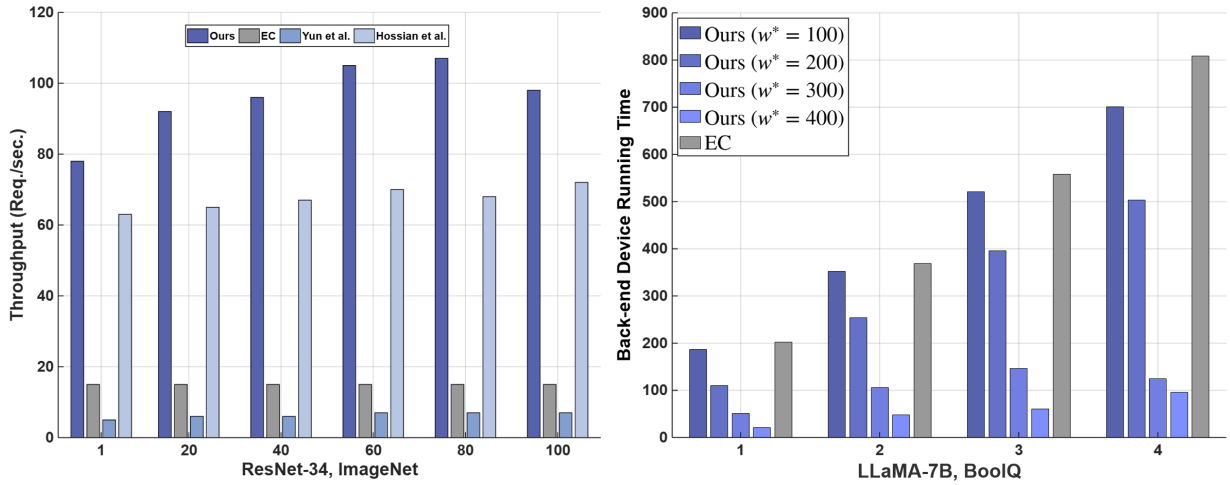

Figure 2: Scalability of our framework in multi-device scenario. Left : Single-shot inference showing the back-end throughput for ResNet-34 on ImageNet as the number of front-end devices increases. Right : Cumulative dev_0 running time required to complete the same BoolQ workload with Llama2-7B, compared against the number of front-end devices.

Table 1: (left) Comparison of IF compression methods on ResNet50 (ImageNet). (right) Comparison on LLM lightweight methods on Llama 2-7B. BPP = bits-per-pixel (lower is better); ΔAcc = accuracy change versus the uncompressed baseline.

Method	ΔAcc	**BPP ↓**
Matsubara *et al.*	0.00	1.28
Duan *et al.* (n4)	−0.30	0.70
Duan *et al.* (n8)	−0.63	0.69
Hossain *et al.* (Cfg-1)	0.00	0.75
Hossain *et al.* (Cfg-2)	0.46	0.69
Ahuja *et al.*	0.00	0.09
Ours	−0.07	**0.08**

Method	Q	PIQA	ARC-e	ARC-c	HS	L_c(ms)
Baseline	16	77.09	53.07	41.04	72.16	7352
OmniQuant	6	74.05	51.70	38.77	68.59	2504
Atom	8	76.06	55.13	40.27	71.30	3576
Atom	3	69.86	46.76	34.73	59.30	968
Ours	C1	75.35	54.76	39.76	69.95	**1824**
Ours	C2	71.71	52.19	39.16	67.45	**1032**

4.2 Results and Discussion

Fig. 2 compares our framework against three competing model partitioning (MP) schemes. For vision inference (left), existing methods like Yun et al. [22] and Hossain et al. [40] are limited by bulky intermediate features or restricted split points. This creates a server bottleneck, causing throughput to degrade as more devices are added. In contrast, our framework's fully flexible split point selection enables superior offloading to the front-end, sustaining the highest back-end throughput. This scalability is mirrored in autoregressive inference (right), where our approach reduces the total server (dev_0) running time by up to 4.4× compared to the cloud-only EC baseline.

Table 1 (left) shows that for vision tasks, our method achieves a state-of-the-art trade-off. It compresses ResNet-50 features to just 0.08 BPP—an order of magnitude smaller than most prior work—while losing a negligible 0.07 percentage points of top-1 accuracy, proving its efficiency even in fixed bottleneck architectures.

Table 1 (right) compares our framework against lightweight LLM techniques. On Llama-2-7B, our method cuts latency from 7.35s to 1.03s while staying within 2% of baseline accuracy. In contrast, OmniQuant requires 2.5x more bandwidth for similar accuracy, while Atom's accuracy drops by 5-10% to match our bandwidth. Our approach is thus unique in its ability to drastically reduce communication without sacrificing task performance.

Fig. 3 (left) shows that the overhead from MP and IF compression remains minimal compared to other delay components, and communication overhead is substantially reduced. Moreover, as shown in Fig. 3 (right), our

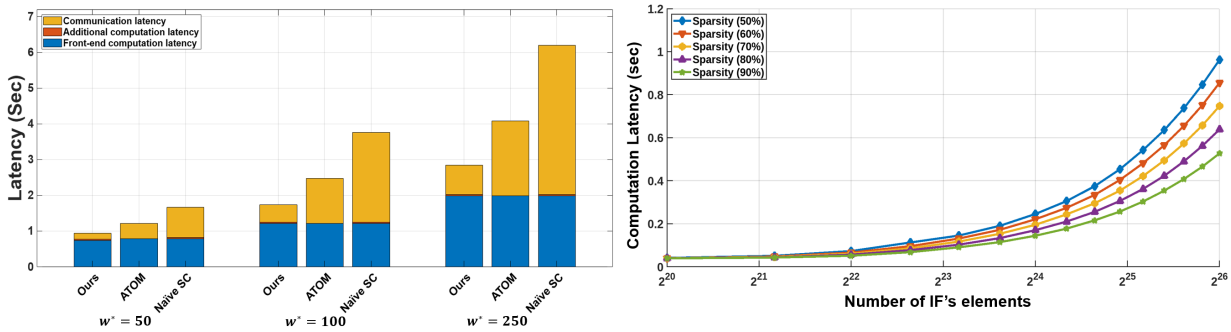

Figure 3: (left) Execution time on the front-end device, IF transmission time, and overhead for the Llama2-7B (BoolQ), comparing different methods across varying device-side computation levels. (right) Computation latency of our framework according to the size of IF.

compression method maintains a nearly constant overhead independent of the actual IF size, validating the efficiency of iterative quantization used in *ABQ*.

Table 2: (left) Single-shot inference. (right) AR inference on diverse models and datasets.

Data	Model	ℓ^*	Sparsity	Acc./mAP
CIFAR-100	ResNet34	2	75%	72.98(-1.46)
ImageNet	ResNet50	3	75%	73.01(-1.54)
	ResNet50	4	90%	74.04(-0.41)
	CaiT-XXS24	30	90%	81.02(-0.94)
GLUE	RoBERTa	2	65%	90.63(+0.03)
	RoBERTa	18	95%	88.56(-2.04)
HellaSwag	Llama2-7B	3	80%	72.60(-2.20)
	Llama2-7B	20	55%	73.19(-0.71)
THUMOS	LSTR	50	55%	69.60(+0.31)
COCO	YOLOv6s	–	95%	0.431(-0.019)

Data	Model	w^*	Sparsity	Acc.
GSM8K (Math)	DeepSeek (Math-7B)	100	70%	81.05(-0.75)
		100	90%	81.20(-0.60)
GSM8K (Math)	DeepSeek (Math-7B)	200	90%	81.65(-0.15)
		200	95%	80.44(-1.36)
GSM8K (Math)	Llama3-8B (Instruct)	100	70%	64.14(-0.61)
		200	90%	63.76(-0.99)
HumanEval (Coding)	Qwen2.5 (Coder-7B)	100	70%	82.32(+0.61)
		200	70%	82.93(+1.22)
HumanEval (Coding)	Llama3-8B (Instruct)	100	70%	54.88(+1.83)
		200	90%	51.83(-1.22)

Table 3: **Multi-modal (text-to-image) generation.**: *FLUX.1-dev* (with Realism LoRA) and *AIDC-AI/Ovis-U1-3B*. **Baseline** offloads the entire generation to the back-end with no partitioning/compression. **Ours** applies our method at diffusion timestep t with sparsity $s \in \{0.6, 0.8\}$. Quantization bit-width $Q = [8, 8, 8]$; ABQ threshold $\delta = 0.2$.

FLUX.1-dev + RealismLoRA

Sparsity	Timestep t	PSNR ↑	SSIM ↑	CLIP-IQA ↑
EC (Baseline)	–	16.61 ± 4.20	0.76 ± 0.12	0.76 ± 0.17
0.6	50	16.91 ± 4.19	0.76 ± 0.12	0.77 ± 0.15
0.8	50	16.62 ± 4.24	0.76 ± 0.12	0.76 ± 0.15
0.6	20	16.81 ± 4.05	0.76 ± 0.12	0.76 ± 0.17
0.8	20	16.93 ± 4.16	0.76 ± 0.12	0.77 ± 0.15

AIDC-AI/Ovis-U1-3B

Sparsity	Timestep t	PSNR ↑	SSIM ↑	CLIP-IQA ↑
EC (Baseline)	–	33.73 ± 4.72	0.98 ± 0.01	0.92 ± 0.03
0.6	50	39.28 ± 4.10	0.99 ± 0.01	0.92 ± 0.03
0.8	50	36.19 ± 3.29	0.98 ± 0.01	0.92 ± 0.03
0.6	20	34.55 ± 5.12	0.98 ± 0.01	0.92 ± 0.03
0.8	20	28.43 ± 5.77	0.96 ± 0.02	0.92 ± 0.02

5 Conclusions

We introduced a novel model partitioning framework to mitigate communication and server-side bottlenecks in deploying large-scale DNNs on multi device setup. By selectively discarding low-magnitude activations and assigning appropriate bitwidths to larger ones, we substantially reduce bandwidth usage with minimal impact on model accuracy. Experimental results on various benchmarks confirm that our framework consistently lowers E2E latency and scales well under multi-device concurrency. Interestingly, its lightweight design does not necessitate retraining or structural modifications, enabling seamless integration with a wide spectrum of DNN architectures, including AR inference in LLMs. Our framework provides a versatile platform for distributed inference, enabling scalable, adaptive, and efficient AI deployments in real-world scenarios.

Table 4: **Ablations.** (left) Module ablation under fixed settings ($\ell = 5$, $s = 0.75$). (mid) Bit allocation via MS. (right) ABQ bit savings by δ.

ATKF −			
MS	ABQ	Acc.	L_c (ms)
−	−	74.53	9592
+	−	74.54	7928
−	+	74.54	8720
+	+	74.58	7152
ATKF +			
MS	ABQ	Acc.	L_c (ms)
−	−	73.02	4736
+	−	73.03	3640
−	+	73.01	4304
+	+	73.01	**3312**

MS			
M	Q	Acc.	L_c (ms)
1	[8]	73.02	3696
1	[8,8]	73.04	4024
2	[8,2]	73.05	3232
3	[4,4,4]	72.98	3328
3	[4,1,1]	72.98	2840
3	[1,4,4]	65.08	3176
3	[1,4,1]	65.08	2928
3	[1,1,4]	68.10	2928
4	[8,4,2,1]	73.01	3080
4	[1,2,4,8]	68.08	3184

Sparsity = 65%		
δ	Acc. (%)	$Avg.Q$
0.05	73.20	[4.0, 7.1, 7.2]
0.10	73.21	[4.0, 7.1, 6.8]
0.15	73.17	[4.0, 6.2, 6.3]
0.20	73.19	[4.0, 5.7, 5.3]
Sparsity = 85%		
δ	Acc. (%)	$Avg.Q$
0.05	69.80	[4.0, 7.0, 7.0]
0.10	69.80	[4.0, 7.0, 7.0]
0.15	69.80	[4.0, 6.9, 6.9]
0.20	69.82	[4.0, 5.9, 5.8]

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 25, 2012.

[3] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 32–42, 2021.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

[5] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces, 2021.

[6] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2023.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 1877–1901, 2020.

[8] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models, 2022.

[9] Yoshitomo Matsubara, Marco Levorato, and Francesco Restuccia. Split computing and early exiting for deep learning applications: Survey and research challenges. *ACM Computing Surveys*, 55(5):1–30, 2022.

[10] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News*, 45(1):615–629, 2017.

[11] Arian Bakhtiarnia, Nemanja Milošević, Qi Zhang, Dragana Bajović, and Alexandros Iosifidis. Dynamic split computing for efficient deep edge intelligence. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2023.

[12] Yoshitomo Matsubara, Davide Callegaro, Sameer Singh, Marco Levorato, and Francesco Restuccia. Bottlefit: Learning compressed representations in deep neural networks for effective and efficient split computing. In *2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 337–346. IEEE, 2022.

[13] Seung-Yeon Kim and Haneul Ko. Distributed split computing system in cooperative internet of things (iot). *IEEE Access*, 2023.

[14] Daniele Jahier Pagliari, Roberta Chiaro, Enrico Macii, and Massimo Poncino. Crime: Input-dependent collaborative inference for recurrent neural networks. *IEEE Transactions on Computers*, 70(10):1626–1639, 2020.

[15] Pratyush Patel, Esha Choukse, Chaojie Zhang, Íñigo Goiri, Brijesh Warrier, Nithish Mahalingam, and Ricardo Bianchini. Characterizing power management opportunities for llms in the cloud. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Volume 3*, pages 207–222, 2024.

[16] Mingyu Sung, Vikas Palakonda, Il-Min Kim, Sangseok Yun, and Jae-Mo Kang. Deco-mesc: Deep compression-based memory-constrained split computing framework for cooperative inference of neural network. *IEEE Transactions on Vehicular Technology*, 2025.

[17] Nilesh Ahuja, Parual Datta, Bhavya Kanzariya, V Srinivasa Somayazulu, and Omesh Tickoo. Neural rate estimator and unsupervised learning for efficient distributed image analytics in split-dnn models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2022–2030, 2023.

[18] Alireza Furtuanpey, Philipp Raith, and Schahram Dustdar. Frankensplit: Efficient neural feature compression with shallow variational bottleneck injection for mobile edge computing. *IEEE Transactions on Mobile Computing*, 2024.

[19] Yoshitomo Matsubara, Ruihan Yang, Marco Levorato, and Stephan Mandt. Supervised compression for resource-constrained edge computing systems. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 2685–2695, 2022.

[20] Zhihao Duan and Fengqing Zhu. Efficient feature compression for edge-cloud systems. In *2022 Picture Coding Symposium (PCS)*, pages 187–191. IEEE, 2022.

[21] Parual Datta, Nilesh Ahuja, V Srinivasa Somayazulu, and Omesh Tickoo. A low-complexity approach to rate-distortion optimized variable bit-rate compression for split dnn computing. In *2022 26th International Conference on Pattern Recognition (ICPR)*, pages 182–188. IEEE, 2022.

[22] Sangseok Yun, Wan Choi, and Il-Min Kim. Cooperative inference of dnns for delay- and memory-constrained wireless iot systems. *IEEE Internet of Things Journal*, 9(17):16113–16127, 2022.

[23] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, 2018.

[24] Mingze Xu, Yuanjun Xiong, Hao Chen, Xinyu Li, Wei Xia, Zhuowen Tu, and Stefano Soatto. Long short-term transformer for online action detection. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 1086–1099, 2021.

[25] Chuyi Li, Lulu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, Liang Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, et al. YOLOv6: A single-stage object detection framework for industrial applications, 2022.

[26] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[27] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255. IEEE, 2009.

[28] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision (ECCV)*, pages 740–755. Springer, 2014.

[29] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models, 2023.

[30] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized bert pretraining approach. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[31] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report, 2024.

[32] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models, 2024.

[33] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, 2018.

[34] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 4791–4800, 2019.

[35] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-20)*, 2020.

[36] Peter Clark, Will Cowhey, Oren Etzioni, Tushar Khot, Daniel Khashabi, and Ashish Sabharwal. Think you have solved question answering? try ARC, the AI2 reasoning challenge, 2018.

[37] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems, 2021.

[38] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code, 2021.

[39] Yu-Gang Jiang, Jingen Liu, Amir Roshan Zamir, George Toderici, Ivan Laptev, Mubarak Shah, and Rahul Sukthankar. THUMOS challenge: Action recognition with a large number of classes. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, 2014.

[40] Md Adnan Faisal Hossain, Zhihao Duan, Yuning Huang, and Fengqing Zhu. Flexible variable-rate image feature compression for edge-cloud systems. In *Proceedings of the 2023 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, pages 182–187, 2023.