CUPA: UCB-Guided Bi-Level Coordination for Edge-Cloud Collaborative ViT Inference

Jiachi Wu[1], Ning Chen[1(✉)], Yanni Xing[2], Hao Pan[1], and He Huang[1]

[1] School of Computer Science and Technology, Soochow University, Suzhou, China
[2] School of Rail Transportation, Soochow University, Suzhou, China
{jcwujcwu,ynxing02,hpanpanhao}@stu.suda.edu.cn,{ningc,huangh}@suda.edu.cn

Abstract. Vision Transformer (ViT) has achieved remarkable performance on a wide range of visual tasks, but its substantial computational cost poses a major challenge to efficient deployment on edge devices. Unlike convolutional neural networks (CNNs), ViT maintains nearly constant intermediate tensor sizes across layers, which limits the effectiveness of partition-only collaborative inference. Moreover, fluctuating network bandwidth directly affects transmission latency, causing the optimal collaboration strategy to vary over time. To address these challenges, we propose CUPA, an edge-cloud collaborative inference framework for ViT in dynamic bandwidth environments. CUPA adopts a hierarchical design. At the outer level, a bandwidth-state-aware upper confidence bound (UCB) algorithm jointly selects the model partition point and the overall pruning ratio for the edge segment in an online manner. At the inner level, CUPA allocates the pruning budget across edge-side layers by jointly considering computational latency benefit and layer-wise sensitivity, thereby refining coarse-grained segment-level decisions into executable layer-wise token pruning schedules. Experimental results show that CUPA reduces average latency by 3.6%–20.5% compared with baseline methods while maintaining competitive Top-1 accuracy.

Keywords: Vision Transformer · Edge-cloud collaboration · Token pruning · UCB · Bandwidth-state awareness

1 Introduction

As vision applications are increasingly deployed on resource-constrained edge devices, improving inference efficiency while preserving accuracy has become a key challenge in edge intelligence [22]. Recently, ViT has achieved outstanding performance on various visual tasks, especially image classification [8, 20], but its large model size and high computational complexity make full on-device inference difficult [11]. Directly uploading raw inputs to the cloud also incurs high communication overhead. Edge-cloud collaborative inference offers a practical solution by partitioning the model between the edge and the cloud.

For CNNs, intermediate feature maps are progressively compressed by downsampling, so a suitable partition point can often strike an effective balance between edge-side computation and communication overhead [9, 13, 15]. For ViT models, however, model partitioning alone is often insufficient, since the token

scale and intermediate representation size remain nearly unchanged across layers, making it difficult to substantially reduce communication latency by only adjusting the partition point.

To alleviate this limitation, existing studies have explored layer-wise token pruning to reduce the size of edge-side intermediate features, thereby lowering transmission overhead and improving overall inference efficiency [1, 16, 18]. However, although token pruning can reduce latency, it also directly affects final prediction accuracy. Therefore, determining how to allocate the pruning budget across layers according to their characteristics becomes a key challenge in edge-cloud collaborative inference for ViT. In addition, real-world bandwidth is inherently dynamic, and the optimal collaboration strategy may vary significantly across bandwidth conditions. As a result, fixed policies often fail to maintain strong performance under fluctuating network conditions, making adaptivity essential for practical edge-cloud ViT inference.

To address these challenges, we propose CUPA, short for Coordinated UCB-Guided Model Partitioning and Token Pruning Allocation, a hierarchical edge-cloud collaborative inference framework for ViT under dynamic bandwidth conditions. CUPA first filters candidate partition-pruning actions based on accuracy and latency constraints, and then constructs safe action sets for different bandwidth states. During online inference, CUPA adopts a two-level decision mechanism: the outer level uses state-wise UCB to select the partition point and overall edge-side pruning ratio according to the current bandwidth, while the inner level allocates the pruning budget across edge-side layers by jointly considering the computational latency benefit and layer sensitivity. This design enables adaptive decision-making in dynamic network environments with low online overhead and achieves a better latency-accuracy trade-off.

We implement and evaluate CUPA on a real-world testbed with a Jetson Orin Nano as the edge device, a GPU-equipped laptop as the cloud server, and real bandwidth traces. Experimental results show that CUPA reduces average latency by 3.6%–20.5% while maintaining competitive Top-1 accuracy.

The main contributions of this paper are summarized as follows:

- We propose a UCB-based online action selection method, which adaptively determines the model partition point and the overall pruning ratio of the edge segment under different bandwidth states.
- We design a layer-wise token pruning allocation method for the edge segment by jointly considering computational latency and layer sensitivity, refining segment-level pruning configurations into executable layer-level decisions.
- Extensive experiments under dynamic bandwidth conditions demonstrate the effectiveness of the proposed method and show that CUPA achieves a better latency-accuracy trade-off than existing approaches.

2 Related Work

Edge-cloud collaborative inference has emerged as an important paradigm for deploying deep models on resource-constrained devices [17, 19]. Representative works such as Neurosurgeon [13], Edgent [15] and JointDNN [9] have shown that collaborative inference can effectively balance local computation and transmission overhead. Recent edge-assisted visual analytics systems have also explored

adaptive configuration, data reduction, and collaborative offloading for efficient video analytics [2, 5, 6] and super-resolution [3, 4]. However, these methods are mainly designed for CNNs or general DNNs and rely on the progressive shrinkage of intermediate feature maps, which does not naturally hold for ViT.

Recent studies have explored Transformer-oriented collaborative inference. DeViT [21] decomposes a large ViT into multiple smaller models for collaborative inference across edge devices, whereas Janus [12] jointly combines model partitioning with token pruning under dynamic network conditions. These studies indicate that ViT collaborative inference depends not only on the partition position, but also on the token budget and network conditions.

Efficient ViT inference has also been extensively studied from the perspective of token pruning. Representative methods include DynamicViT [18], EViT [16] and ToMe [1]. Although these methods are effective for accelerating ViT inference on a standalone model or device, edge-cloud collaborative settings require token compression to be jointly considered with model partitioning.

3 System Model

3.1 Edge-Cloud Inference Architecture

We consider an edge-cloud collaborative inference system composed of an edge device D and a cloud server S, whose computational capacities are denoted by C_D and C_S, respectively. The two devices are connected through a wireless network with available bandwidth B. Given an input image, the initial layers of the Transformer model are executed on the edge device D, while the remaining layers are offloaded to the cloud server S for further processing.

We consider a ViT model with L sequential layers for edge-cloud collaborative inference, indexed by $\mathcal{L} = \{1, 2, \ldots, L\}$. To improve inference efficiency, we introduce two key decision variables: the partition point v and the edge pruning ratio r. Specifically, the partition point $v \in \{1, 2, \ldots, L-1\}$ indicates that the first v layers are executed on the edge device, whereas the remaining layers $(v+1, \ldots, L)$ are offloaded to the cloud server. In this work, token pruning is applied only on the edge side, because reducing edge computation and the number of transmitted intermediate tokens is more critical to end-to-end latency, while the cloud server is relatively resource-rich. The pruning ratio r is selected from a predefined discrete set $\mathcal{R} = \{r_1, r_2, \ldots, r_K\}$, where each value represents the overall proportion of tokens to be removed on the edge device and K denotes the number of candidate pruning ratios. By adjusting r, the system controls the number of tokens retained after edge inference.

Let N_0 denote the number of tokens generated by the input embedding. During edge inference, token pruning is progressively applied to reduce the number of tokens, thereby decreasing both computation and communication overhead. Let n_l denote the number of tokens entering layer l, where $n_1 = N_0$. For each edge layer $l \leq v$, p_l tokens are pruned before executing that layer. Let m_l denote the number of retained tokens actually processed by layer l. Then

$$m_l = n_l - p_l, \quad n_{l+1} = m_l. \tag{1}$$

After edge inference, the remaining tokens n_{v+1} are transmitted to the cloud server for subsequent processing.

3.2 Accuracy Model

Token pruning may degrade model accuracy, and different layers exhibit different levels of sensitivity to pruning. In CUPA, inference accuracy is influenced by two decision variables: the partition point v and the edge pruning ratio r.

Let $Acc(v, r)$ denote the inference accuracy under partition point v and pruning ratio r. Due to the complex interactions among tokens and Transformer layers, it is difficult to derive an explicit analytical form for the accuracy. Hence, we treat the accuracy model as a black-box function evaluated on validation data,

$$Acc(v, r) = f(v, r), \tag{2}$$

where $f(\cdot)$ characterizes the empirical relationship between the decision variables and the resulting model accuracy.

In practice, the system evaluates the inference accuracy under each configuration (v, r) and uses the observed accuracy as feedback in the reward signal. This feedback-driven design allows the framework to guide subsequent decision-making without requiring an explicit analytical accuracy model.

3.3 Latency Model

The total inference latency consists of three components, i.e., edge computation latency, transmission latency, and cloud computation latency.

Edge Computation Latency. For each ViT layer, the computation mainly arises from the self-attention module and the MLP block. Let d denote the token feature dimension and γ denote the MLP expansion ratio. Based on the token number m_l, the computational cost of layer l can be approximated as

$$F_l(m_l) \approx (4 + 2\gamma)m_l d^2 + 2m_l^2 d, \tag{3}$$

where two terms correspond to projection/MLP and attention costs, respectively.

Accordingly, the edge computation latency can be expressed as

$$T_{edge} = \sum\nolimits_{l=1}^{v} \frac{F_l(m_l)}{C_D}. \tag{4}$$

Transmission Latency. After edge inference, the remaining n_{v+1} tokens are transmitted to the cloud server for further processing. The transmission latency can be expressed as

$$T_{trans} = \frac{\rho n_{v+1} d s}{B}, \tag{5}$$

where s denotes the data size of each feature element (in bytes) and ρ represents the compression ratio applied during transmission.

Cloud Computation Latency. After transmission, the remaining layers are executed on the cloud server using the transmitted tokens. Since the token number remains unchanged during cloud inference, the cloud computation latency can be expressed as

$$T_{cloud} = \sum\nolimits_{l=v+1}^{L} \frac{F_l(n_{v+1})}{C_S}. \tag{6}$$

Combining the above components, the total inference latency is given by

$$T_{total} = T_{edge} + T_{trans} + T_{cloud}. \tag{7}$$

3.4 Problem Formulation

Based on the above models, the optimization problem is to jointly determine the partition point v and the edge pruning ratio r so as to minimize total latency while satisfying the accuracy requirement. For a given (v, r), the pruning budget is allocated across the first v layers, resulting in a layer-wise pruning vector $\{p_l\}_{l=1}^{v}$ determined by the internal pruning budget allocation strategy, and the total latency is denoted by $T_{total}(v, r)$. The problem can be formulated as

$$\mathcal{P}: \quad \min_{v,r} \; T_{total}(v, r) \tag{8}$$

$$\text{s.t.} \quad \text{C1:} \;\; Acc(v, r) \geq Acc_{min}, \tag{9}$$

$$\text{C2:} \;\; v \in \{1, 2, \ldots, L-1\}, \quad r \in \mathcal{R}, \tag{10}$$

$$\text{C3:} \;\; \sum\nolimits_{l=1}^{v} p_l = \lfloor rN_0 \rfloor, \tag{11}$$

$$\text{C4:} \;\; 0 \leq p_l \leq \lfloor \frac{n_l}{2} \rfloor, \quad \forall l \leq v, \tag{12}$$

where Acc_{min} denotes the minimum acceptable inference accuracy.

Constraint C1 ensures that the selected collaborative inference configuration satisfies the required accuracy. C2 specifies the feasible domains of the partition point and the pruning ratio. C3 guarantees that the total number of pruned tokens is consistent with the pruning budget determined by r. C4 ensures that the number of pruned tokens at each edge layer does not exceed half of the available tokens, which is consistent with the ToMe-based token-pruning constraint. Since the decision variables are discrete and the accuracy of each candidate configuration is only available through empirical evaluation, the formulated problem is a constrained combinatorial optimization problem that is difficult to solve using conventional optimization methods.

4 CUPA Design

4.1 Overview of CUPA

CUPA consists of an offline profiling stage and an online decision stage, as shown in Fig. 1. In the offline stage, candidate actions are evaluated on a validation dataset to obtain their accuracy, and their latency profiles are measured under a fixed reference bandwidth. Based on the minimum accuracy requirement, infeasible actions are discarded to construct a candidate action set. Meanwhile, a layer-wise sensitivity table is built to characterize the impact of token pruning on different edge-side layers and support intra-segment token-pruning allocation.

Based on the offline profiling results, CUPA performs online decision-making through a two-level framework. At the outer level, a state-aware UCB policy determines the partition point and the total token-pruning ratio on the edge side according to the current network condition. At the inner level, an allocator distributes the assigned pruning budget across the edge-side layers by jointly considering layer-wise computational latency benefit and sensitivity, and the resulting token pruning at each layer is implemented using ToMe. The details of these components are presented in the following subsections.

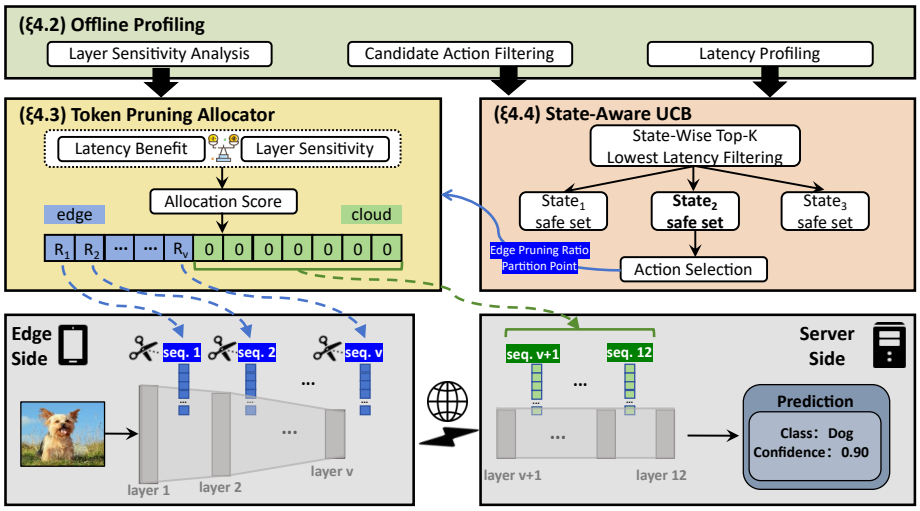


Fig. 1: Overview of CUPA

4.2 Offline Profiling

The offline profiling stage provides the information required for subsequent online decision-making and token-pruning allocation. Specifically, it includes candidate action construction, latency profile collection, and layer-wise sensitivity analysis.

First, each action $a = (v, r)$ is evaluated on a validation dataset to obtain its accuracy. To satisfy the minimum accuracy requirement Acc_{min}, actions whose accuracy fall below the threshold are discarded. The remaining actions are retained as candidates for subsequent decision-making. This ensures the accuracy requirement while reducing the UCB action space.

Second, the total latency of each candidate action is recorded under a fixed reference bandwidth and later used to refine the online UCB action set.

Third, we analyze layer-wise sensitivity to token pruning. For each layer $l \in \mathcal{L}$, we apply single-layer ToMe pruning while keeping other layers unchanged, and evaluate the Top-1 accuracy under compression rates $\hat{r} \in \hat{R}$. Fig. 2 presents the Top-1 accuracy of different layers under varying compression rates. The results show that layers respond quite differently to token pruning: as the compression rate increases, some shallow layers exhibit a much faster drop in accuracy, whereas deeper layers remain relatively stable. This observation indicates that shallow layers tend to be more sensitive to token pruning, especially at larger compression rates. Let $Acc_l(\hat{r})$ denote the accuracy when only layer l is pruned with compression rate $\hat{r}$, and Acc_{base} denote the accuracy without token pruning. We define the non-negative accuracy drop of layer l at compression rate $\hat{r}$ as

$$\Delta Acc_l(\hat{r}) = \max(Acc_{base} - Acc_l(\hat{r}), 0). \tag{13}$$

Based on the measured accuracy-drop curve, the initial sensitivity of layer l is computed as the area under the curve, i.e.,

$$S_l = \int_0^{\hat{r}_{max}} \Delta Acc_l(\hat{r})\, d\hat{r}, \tag{14}$$

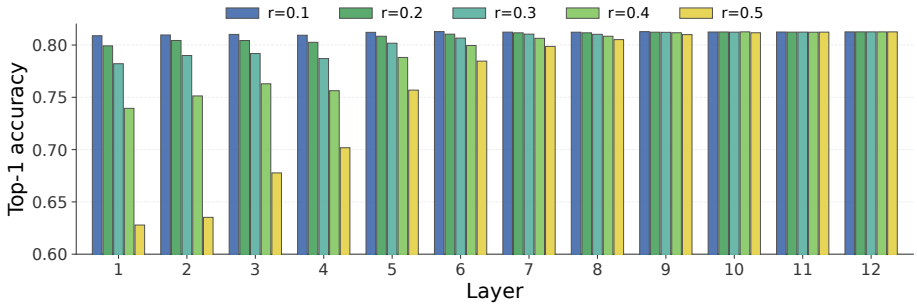


Fig. 2: Top-1 accuracy under single-layer pruning

which is implemented in practice using trapezoidal numerical integration over the sampled compression rates. A larger S_l indicates that pruning layer l causes more severe accuracy degradation and thus implies higher sensitivity to token pruning. The resulting sensitivity table is later used as a prior for layer-wise token-pruning allocation to avoid overly aggressive pruning on sensitive layers.

Offline profiling is a one-time cost for a fixed model-task setting and only needs to be repeated when the model architecture or target task changes.

4.3 Layer-wise Token Pruning Allocator

CUPA employs a layer-wise allocator to distribute the token-pruning budget across the selected edge-side layers. For a given partition point v and pruning ratio r, the total budget $\lfloor rN_0 \rfloor$ is allocated across the first v layers to obtain the layer-wise pruning vector $\{p_l\}_{l=1}^{v}$. The allocator jointly considers the edge-side computational latency benefit and the sensitivity of each layer. Specifically, for each edge-side layer $l \leq v$, CUPA estimates its unit latency benefit and combines it with the offline sensitivity measurement to compute the allocation score, i.e.,

$$\psi_l = \tilde{b}_l e^{-\lambda \tilde{S}_l}, \tag{15}$$

where $\tilde{b}_l$ and $\tilde{S}_l$ denote the normalized unit latency benefit and normalized sensitivity of layer l, respectively, and λ controls the sensitivity penalty. Thus, layers with higher latency benefit and lower sensitivity receive larger pruning weights.

The scores are then normalized into allocation weights and converted into integer layer-wise token-pruning amounts. The resulting allocation is further adjusted to satisfy the feasibility constraints of ToMe pruning, yielding an executable pruning schedule for the edge-side layers. In this way, CUPA refines the segment-level configuration into fine-grained layer-wise token-pruning decisions while balancing latency reduction against accuracy preservation.

4.4 Online Decision with State-Aware UCB

In the online stage, CUPA adopts a state-aware UCB policy [14] for action selection under dynamic bandwidth conditions. At step t, the current bandwidth B_t is first observed and mapped to a bandwidth state $s_t \in \{\text{low,middle,high}\}$. Let

$\mathcal{A}$ denote the candidate action set obtained in the offline stage. Because infeasible actions that violate the accuracy requirement have already been removed offline, the online policy only needs to select a latency-efficient action from $\mathcal{A}$.

To reduce online search complexity, CUPA further exploits the offline latency profiles to perform state-specific candidate action filtering. For each state s, the latency of action $a \in \mathcal{A}$ is estimated under a state-dependent reference bandwidth, denoted by $\hat{T}_s(a)$. CUPA then ranks all candidate actions according to $\hat{T}_s(a)$ and retains only the top-K actions with the smallest estimated latency to form the safe action set $\mathcal{A}_s^{\text{safe}}$. In this way, online exploration is restricted to actions that are more likely to match the current network state, thereby improving both decision efficiency and robustness.

For each state s, CUPA maintains independent UCB statistics. Let $N_s(a)$ denote the number of times action a is selected in state s, and let $Q_s(a)$ denote its value estimate. At round t, the UCB score is computed as

$$U_s(a) = Q_s(a) + c_t \sqrt{\frac{\ln(\max(1, \tau_s))}{N_s(a) + \epsilon}}, \tag{16}$$

where τ_s is the number of decision rounds observed in state s, c_t is the exploration coefficient, and ϵ is a small constant for numerical stability. CUPA then selects the action with the largest score from $\mathcal{A}_s^{\text{safe}}$. By maintaining separate statistics for different bandwidth states, CUPA adapts action selection to varying network conditions in a state-aware manner. After an action is selected, the allocator in Section 4.3 is invoked to distribute the corresponding token-pruning budget across the edge-side layers and produce a layer-wise pruning schedule.

5 Evaluations

5.1 Experimental Setup

Hardware. We use an NVIDIA Jetson Orin Nano as the edge device to execute the front layers and a laptop equipped with an NVIDIA GeForce RTX 5060 GPU as the cloud-side device for the remaining layers.

Network Trace Dataset. We use an open-source bandwidth trace dataset collected from real cellular networks [10] to emulate dynamic communication conditions between the edge and the cloud.

Tasks and Evaluation Metrics. We use DeiT-Base [20] on ImageNet-1K dataset [7] for edge-cloud collaborative image classification, and evaluate CUPA by Top-1 accuracy, average end-to-end latency, and latency CDF.

Parameter Settings. We initialize the exploration coefficient c_t to 1.0, which gradually decays with the number of online rounds to a lower bound of 0.1, and set the sensitivity penalty coefficient λ to 2.0 for the experiments. For Top-K filtering, we retain the top $K = 5$ candidate actions with the lowest estimated latency for each bandwidth state to construct the corresponding safe action set. The candidate pruning ratio set is $\mathcal{R} = \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5\}$.

Baselines. We compare CUPA with four baselines:

- Global-UCB: In contrast to CUPA, Global-UCB is an online decision method without bandwidth-state partitioning.

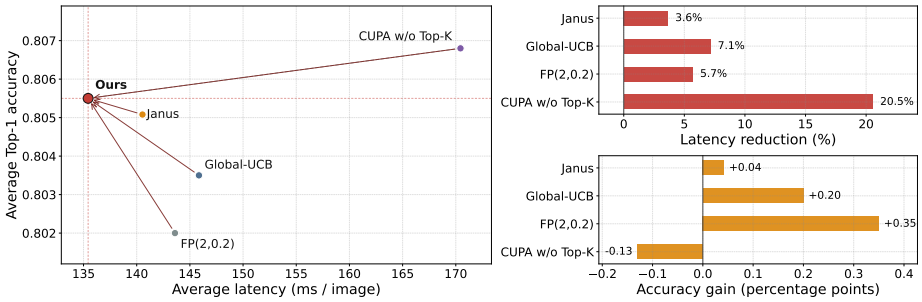
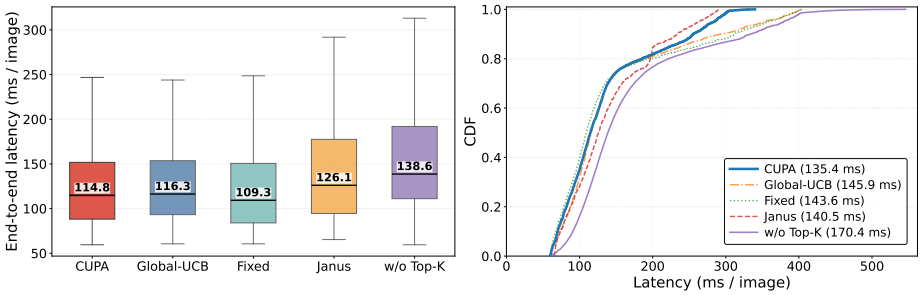


Fig. 3: Performance differences between CUPA and baselines



(a) Latency distribution

(b) Latency CDF

Fig. 4: Latency distribution comparison of different methods

– Fixed Policy (FP): This method performs inference with a fixed partition point (e.g., 2) and a fixed edge-side pruning ratio (e.g., 0.2), without adapting the decision to bandwidth variations.
– CUPA without Top-K Filtering (CUPA w/o Top-K): This variant adopts bandwidth-state partitioning and state-aware UCB decision-making, but does not apply Top-K lowest-latency filtering.
– Janus [12]: Janus is a collaborative edge-cloud inference method that combines token pruning and model partitioning, and adaptively selects the pruning level and partition point according to network conditions.

5.2 Baseline Comparison

Fig. 3 presents the comparison between CUPA and the baselines. CUPA achieves the best overall latency-accuracy trade-off, outperforming Janus, Global-UCB, and FP$(2, 0.2)$ with 3.6%, 7.1% and 5.7% lower average latency, respectively, while also improving Top-1 accuracy by 0.04, 0.20 and 0.35 percentage points. Compared with CUPA w/o Top-K, CUPA further reduces latency by 20.5% with only a 0.13-point drop in accuracy, confirming the benefit of Top-K filtering.

We further analyze the latency distributions of different methods and observe that CUPA provides not only lower average latency but also more stable latency behavior. As shown in Fig. 4(a), CUPA exhibits a relatively low median latency and a more compact distribution, indicating a lower overall latency level and

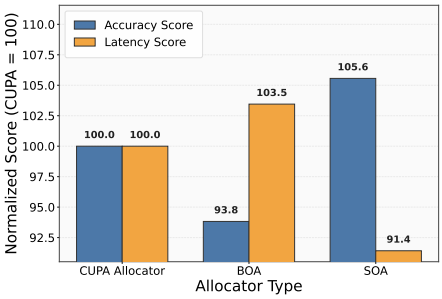


Fig. 5: Allocator comparison in accuracy and latency

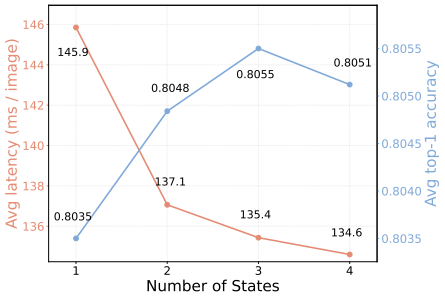
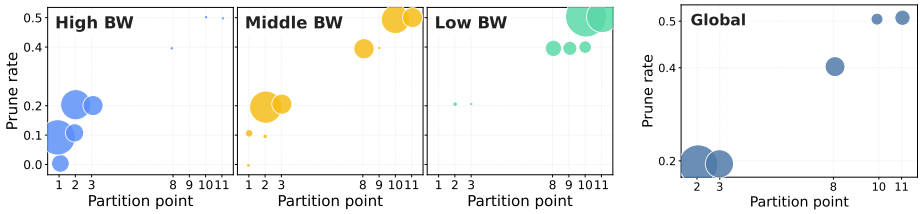


Fig. 6: Impact of bandwidth state number on latency and accuracy



(a) CUPA (b) Global-UCB

Fig. 7: Action distribution comparison between CUPA and Global-UCB

better stability. In comparison, Janus and especially CUPA w/o Top-K show higher medians and more dispersed latency distributions, indicating that these methods are more prone to high-latency cases. Fig. 4(b) further presents the cumulative distribution function (CDF) of end-to-end latency. The CUPA curve is overall shifted to the left relative to most baselines, indicating that lower latency is achieved for a larger fraction of samples.

To evaluate the layer-wise pruning allocator that jointly considers layer sensitivity and computational latency benefit, we compare the proposed allocator with the benefit-only allocator (BOA) and the sensitivity-only allocator (SOA). As shown in Fig. 5, taking the proposed CUPA allocator as the baseline (score=100), BOA obtains a latency score of 103.5 but only 93.8 in accuracy score. In contrast, SOA reaches 105.6 in accuracy score, while its latency score drops to 91.4. This indicates that BOA is more latency-oriented, whereas SOA is more accuracy-oriented. By jointly considering layer sensitivity and computational latency benefit, the proposed allocator achieves a more balanced trade-off between latency reduction and accuracy preservation.

We further examine the impact of the number of bandwidth states. As shown in Fig. 6, increasing the number of states from 1 to 3 reduces average latency from 145.9 ms to 135.4 ms and improves average Top-1 accuracy from 0.8035 to 0.8055. However, further increasing it to 4 yields only marginal latency improvement while slightly reducing accuracy. Therefore, we adopt 3 states in CUPA.

Finally, Fig. 7 shows that CUPA learns distinct action preferences under different bandwidth states. Under high bandwidth, the selected actions are mainly concentrated on earlier partition points with relatively lower pruning rates. Un-

der medium bandwidth, the action choices become more balanced. Under low bandwidth, the selected actions shift toward deeper edge execution and more aggressive pruning. In contrast, Global-UCB mainly relies on a few globally frequent actions without differentiating among bandwidth states. This result indicates that CUPA can better adapt its decisions to varying network conditions, which explains its superior latency–accuracy performance.

6 Conclusion

In this paper, we investigate edge-cloud ViT collaborative inference under dynamic bandwidth conditions and propose CUPA, a hierarchical collaborative inference framework. By combining a state-aware UCB decision mechanism with a layer-wise pruning allocation strategy that jointly considers computational latency benefit and layer sensitivity, CUPA adaptively determines the partition point and the overall pruning ratio of the edge segment, and further refines coarse-grained collaborative decisions into executable layer-wise pruning schedules. Experimental results show that CUPA achieves a superior latency-accuracy trade-off under dynamic network conditions, effectively reducing average inference latency while maintaining competitive Top-1 accuracy. These results verify the effectiveness of the proposed method for adaptive edge-cloud ViT inference.

Acknowledgments. The corresponding author is Ning Chen (ningc@suda.edu.cn). This work was supported in part by NSFC (Grant No. 62502331 and No. 62332013), Natural Science Foundation of Jiangsu Province (Grant No. BK20250783), China Postdoctoral Science Foundation (Grant No. 2025M771496), State Key Lab. for Novel Software Technology of Nanjing University (Grant No. KFKT2025B24), Jiangsu Funding Program for Excellent Postdoctoral Talent.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Bolya, D., Fu, C.Y., Dai, X., Zhang, P., Feichtenhofer, C., Hoffman, J.: Token merging: Your vit but faster. arXiv preprint arXiv:2210.09461 (2022)
2. Chen, N., Quan, S., Zhang, S., Qian, Z., Jin, Y., Wu, J., Li, W., Lu, S.: Cuttlefish: Neural configuration adaptation for video analysis in live augmented reality. IEEE Transactions on Parallel and Distributed Systems **32**(4), 830–841 (2020)
3. Chen, N., Zhang, S., Liang, Y., Wu, J., Chen, Y., Yan, Y., Qian, Z., Lu, S.: Tilesr: Accelerate on-device super-resolution with parallel offloading in tile granularity. In: IEEE INFOCOM 2024-IEEE Conference on Computer Communications. pp. 2538–2547. IEEE (2024)
4. Chen, N., Zhang, S., Ma, Z., Chen, Y., Jin, Y., Wu, J., Qian, Z., Liang, Y., Lu, S.: Vichaser: Chase your viewpoint for live video streaming with block-oriented super-resolution. IEEE/ACM Transactions on Networking **32**(1), 445–459 (2023)
5. Chen, N., Zhang, S., Wu, J., Huang, H., Lu, S.: Spliceosome: On-camera video thinning and tuning for timely and accurate analytics. IEEE Transactions on Networking **33**(3), 1252–1264 (2025)

6. Chen, N., Zhang, S., Zhang, S., Yan, Y., Chen, Y., Lu, S.: Resmap: Exploiting sparse residual feature map for accelerating cross-edge video analytics. In: IEEE INFOCOM 2023-IEEE Conference on Computer Communications. pp. 1–10. IEEE (2023)
7. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255. Ieee (2009)
8. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al.: An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929 (2020)
9. Eshratifar, A.E., Abrishami, M.S., Pedram, M.: Jointdnn: An efficient training and inference engine for intelligent mobile cloud computing services. IEEE transactions on mobile computing **20**(2), 565–576 (2019)
10. Ghoshal, M., Kong, Z.J., Xu, Q., Lu, Z., Aggarwal, S., Khan, I., Li, Y., Hu, Y.C., Koutsonikolas, D.: An in-depth study of uplink performance of 5g mmwave networks. In: Proceedings of the ACM SIGCOMM Workshop on 5G and Beyond Network Measurements, Modeling, and Use Cases. pp. 29–35 (2022)
11. Hu, C., Li, B.: When the edge meets transformers: Distributed inference with transformer models. In: 2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS). pp. 82–92. IEEE (2024)
12. Jiang, L., Fu, S.D., Zhu, Y., Li, B.: Janus: Collaborative vision transformer under dynamic network environment. In: IEEE INFOCOM 2025-IEEE Conference on Computer Communications. pp. 1–10. IEEE (2025)
13. Kang, Y., Hauswald, J., Gao, C., Rovinski, A., Mudge, T., Mars, J., Tang, L.: Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. ACM SIGARCH Computer Architecture News **45**(1), 615–629 (2017)
14. Lattimore, T., Szepesvári, C.: Bandit algorithms. Cambridge University Press (2020)
15. Li, E., Zhou, Z., Chen, X.: Edge intelligence: On-demand deep learning model co-inference with device-edge synergy. In: Proceedings of the 2018 workshop on mobile edge communications. pp. 31–36 (2018)
16. Liang, Y., Ge, C., Tong, Z., Song, Y., Wang, J., Xie, P.: Not all patches are what you need: Expediting vision transformers via token reorganizations. arXiv preprint arXiv:2202.07800 (2022)
17. Liu, J., Du, Y., Yang, K., Wu, J., Wang, Y., Hu, X., Wang, Z., Liu, Y., Sun, P., Boukerche, A., et al.: Edge-cloud collaborative computing on distributed intelligence and model optimization: A survey. IEEE Communications Surveys & Tutorials (2026)
18. Rao, Y., Zhao, W., Liu, B., Lu, J., Zhou, J., Hsieh, C.J.: Dynamicvit: Efficient vision transformers with dynamic token sparsification. Advances in neural information processing systems **34**, 13937–13949 (2021)
19. Tang, S., Cui, M., Qi, L., Xu, X.: Edge intelligence with distributed processing of dnns: A survey. Computer Modeling in Engineering & Sciences **136**(1), 5 (2023)
20. Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., Jégou, H.: Training data-efficient image transformers & distillation through attention. In: International conference on machine learning. pp. 10347–10357. PMLR (2021)
21. Xu, G., Hao, Z., Luo, Y., Hu, H., An, J., Mao, S.: Devit: Decomposing vision transformers for collaborative inference in edge devices. IEEE Transactions on Mobile Computing **23**(5), 5917–5932 (2023)
22. Zhou, Z., Chen, X., Li, E., Zeng, L., Luo, K., Zhang, J.: Edge intelligence: Paving the last mile of artificial intelligence with edge computing. Proceedings of the IEEE **107**(8), 1738–1762 (2019)