Fast and Accurate Classification with Parallel IDK Classifier Cascades

Ishrak Jahan Ratul
Department of Computer Science
Texas State University
ishrakratul@txstate.edu

Kecheng Yang
Department of Computer Science
Texas State University
yangk@txstate.edu

***Abstract*—Object recognition in edge systems requires a balance between prediction accuracy and strict decision time. Existing approaches like model compression, pruning, quantization, and early exit mechanisms can reduce computation but often require architectural modification of networks, retraining, or specialized deployment support, and most inference pipelines rely on sequential task execution, limiting their ability to exploit the task parallelism capabilities of modern GPUs. The IDK ("I Don't Know") cascade addresses accuracy-decision time trade-offs by routing each input through a sequence of classifiers, enabling simple inputs to be resolved quickly while reserving computation-intensive models for difficult cases. In this paper, we present a parallel IDK Cascade framework that extends the sequential cascade to parallel multistream GPU execution, increasing kernel execution overlap and lowering average decision time without changing model architectures or GPU runtime behavior. Experiments on ImageNet show that the proposed design reduces average inference time significantly while maintaining predictive accuracy.**

***Index Terms*—Adaptive Inference, IDK Cascade, Parallel Execution, Multistream Processing, Object Recognition, Inference Optimization, Confidence Routing**

I. Introduction

In the past decade, neural networks have achieved substantial improvements in predictive performance for object recognition and classification tasks. Enhancing network depth, increasing parameter quantities, and integrating additional architectural components have collectively led to improved classification accuracy [30]. However, these enhancements are accompanied by a corresponding increase in computational demand. High accuracy is seldom free. It comes at the cost of longer inference times, increased memory consumption, and prolonged use of compute resources [21]. As models become more complex, each prediction necessitates additional operations and processing time. A model that requires more time to reach a decision utilizes more computational resources over time, decreasing system efficiency, reducing throughput, and increasing energy consumption [33].

Model compression and the design of lighter architectures are two common responses to this challenge. Pruning, quantization, architecture simplification, and knowledge distillation are all techniques for reducing model size and computational cost [30]. While these approaches are useful, they have their own limitations. Compression techniques may reduce accuracy, necessitate retraining, or rely on precise tuning [33].

This work is supported in part by NSF grant 2442078.

Another line of research explores adaptive inference and early exit mechanisms, where intermediate layers of a deep network are equipped with auxiliary classifiers to enable dynamic termination [11]. Cascaded and multi-exit systems attempt to balance accuracy and inference time by selectively invoking stronger models only when necessary [1, 14]. While these methods improve average efficiency, many still rely on architectural modification, joint training, or tightly coupled model designs. Moreover, their execution cannot leverage available task parallelism, limiting their ability to exploit stream-level concurrency available in modern GPU platforms [16].

The IDK ("I Don't Know") cascade framework [1, 2] represents a promising step toward closing this gap. Rather than creating a new lightweight or smaller network, the cascade arranges multiple existing classifiers from simple to complex [18]. If a classifier's confidence exceeds a threshold, the prediction is accepted; otherwise, the input is routed to a more accurate and computationally intensive model [1, 14]. This way, lightweight models resolve simple inputs quickly while difficult inputs receive more computational attention. Overall accuracy is maintained because difficult inputs are routed to the most accurate classifier when necessary [1, 18]. Despite these advantages, most existing cascade implementations are sequential. Each input passes through stages in strict serial order, which limits hardware utilization [14, 33]. Modern GPUs enable concurrent execution across multiple streams, but traditional cascades do not exploit this capability.

In this paper, we address this gap by introducing a parallel IDK cascade framework. Instead of processing cascade stages in strict order, different classifiers are assigned to separate GPU execution streams [16], allowing computation to overlap across cascade stages [26]. The average inference time decreases not only because fewer inputs reach the heaviest model, but also because multiple stages run concurrently. The framework retains the full benefits of the original IDK cascade: no internal model modifications are required, pretrained classifiers are reused as independent components, and confidence-based decision rules remain unchanged.

Our contributions. In this work, we present an implementation of parallel IDK cascades, a parallel variant of confidence-based IDK cascades that allows classifiers to run concurrently across multiple GPU streams, preserving each classifier's architecture and predictive behavior while achieving higher

end-to-end inference efficiency. We apply a greedy threshold-lowering method that dynamically adjusts model confidence thresholds to favor earlier acceptance, maintaining accuracy targets while maximizing early exits. We design a structured approach for assigning classifiers to parallel execution streams, increasing kernel overlap and GPU utilization without modifying CUDA scheduling mechanisms or affecting runtime semantics. We evaluate the proposed system on ImageNet using an NVIDIA Jetson AGX Orin, where comparisons with single-model inference and sequential IDK cascades under multiple precision targets show consistent accuracy with significantly lower average inference time and improved worst-case decision time.

II. Related Work

Our work revolves around three closely related research areas: selective classification, cascade-based adaptive inference, and parallel execution strategies.

A. Selective Classification and Reject Options

The *reject option* prevents a classifier from making a prediction if confidence falls below a predefined threshold. Recent research revisits these principles within the broader framework of *selective classification*, where models trade coverage for improved reliability [10]. The IDK ("I Don't Know") classifier operationalizes this concept by providing an explicit abstention output when confidence is insufficient [1]. Unlike traditional reject-option systems where rejection terminates inference, IDK outputs serve as cascade routing signals, directing abstained inputs to downstream classifiers for further processing.

B. Out-of-Distribution Detection and Cascading

Out-of-distribution detection seeks to identify inputs that may result in unreliable predictions. Larson et al. [17] found that intent classification systems frequently miss out-of-scope inputs, highlighting limitations in confidence estimation. Subsequent research investigated uncertainty-aware mechanisms to identify ambiguous samples [31]. Cascade architectures address uncertainty by routing low-confidence inputs to more robust models, emphasizing computational efficiency alongside reliability. Sequential cascades defer uncertain cases to stronger classifiers [1], while early exit architectures enable simple samples to terminate inference at intermediate layers [27]. Window-based ensemble cascades further improve uncertainty handling on boundary samples [31].

C. Parallel Execution and Threshold Calibration

Parallel execution strategies were investigated primarily to reduce wall-clock latency. Speculative approaches run multiple candidate models concurrently and verify outputs [25], but increase total computation because multiple models may run unnecessarily. The IDK scheduling framework optimizes average execution time with optional worst-case latency constraints [1], yet most implementations rely on serialized execution, leaving the potential of concurrent computation unexplored. Threshold selection methods include post-hoc calibration [7, 24], joint optimization during training, and Learning-to-Cascade approaches that optimize both classifier loss and routing policies [6]. Window-based methods utilize implicit boundary regions to guide further computation [31]. Although threshold optimization has received considerable attention, little research has examined how routing decisions interact with parallel cascade execution.

Classifier	Top-1 Accuracy (%)	Average Inference Time (ms)
AlexNet	59.57	6.92
ResNet18	71.98	10.36
ConvNeXt	77.10	17.89
EfficientNet	79.22	27.72
Swin Transformer	90.29	66.47

TABLE I: Baseline performance of independently deployed classifiers

III. Baseline Inference

Before introducing the parallel IDK cascade, we first define the baseline inference setting. All inference experiments were performed on the NVIDIA Jetson AGX Orin developer kit, which includes a 2048-core NVIDIA Ampere GPU, 64 Tensor cores, and dual NVDLA accelerators [13], using 50-watt power mode (MODE 50W). PyTorch is used as the software framework for both fine-tuning and inference. We evaluated models on the Tiny ImageNet dataset [19], which includes 200 object classes with 500 training and 50 validation images per class. The original validation set of 10,000 images was held out exclusively for testing, and the training set was divided 90/10 for training and validation to prevent data leakage.

We fine-tuned AlexNet [15], ResNet18 [9], EfficientNet [29], ConvNeXt [23], and Swin Transformer [22] as candidate classifiers. Each model was initialized with ImageNet-pretrained weights and fine-tuned using Distributed Data Parallel training on dual NVIDIA RTX A4000 GPUs. Training used cross-entropy loss via `nn.CrossEntropyLoss()`, optimized with SGD (learning rate 0.001, momentum 0.9) for up to 15 epochs with early stopping on validation loss. Random cropping, horizontal flipping, and color jittering were applied for generalization. The baseline analysis evaluates each standalone model in terms of top-1 accuracy and average inference time per input.

Table I shows the accuracy and average inference time of each model when deployed independently. The results show a consistent trade-off between accuracy and inference time. Light architectures like AlexNet have low average inference time but low predictive accuracy, while Swin Transformer achieves the highest accuracy at significantly longer execution time. Intermediate models such as ResNet18 and EfficientNet provide balanced trade-offs but still show notable increases in inference time as accuracy improves.

In the baseline configuration, every input is processed using the same model regardless of input difficulty, resulting in uniform computation and inefficient use of resources. Accuracy and inference time are tightly coupled, and single model deployment cannot simultaneously optimize both. These constraints motivate threshold-based adaptive cascading strategies that dynamically choose classifiers based on input complexity.

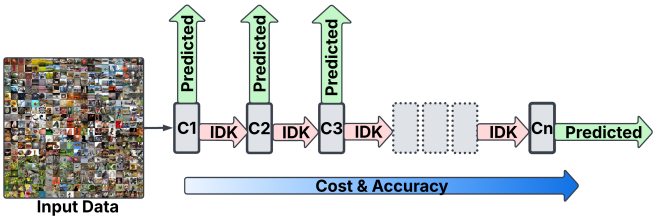

Fig. 1: Illustration of an IDK cascade with classifiers $C_1, C_2, \ldots, C_n$

IV. Sequential IDK Cascade

This section describes how to build a practical IDK cascade using standalone classifiers, covering confidence threshold search, classifier ordering, and deployment strategy.

A. IDK Classifier

An IDK cascade [1, 2] is an adaptive classification framework that processes inputs through an ordered sequence of classifiers. Each classifier either produces a confident prediction or returns an explicit IDK outcome [1], forwarding the input to the next classifier. Figure 1 illustrates the operational principle of an IDK cascade.

Let C_i denote the i-th classifier and x an input sample. If the confidence score $P(C_i(x))$ exceeds a predefined threshold τ_i [6], the classifier outputs the predicted class y_i; otherwise, it returns IDK and forwards the input to the next classifier [1]. Formally,

$$\text{Output of } C_i(x) = \begin{cases} y_i, & \text{if } P(C_i(x)) \geq \tau_i, \\ \text{IDK}, & \text{if } P(C_i(x)) < \tau_i. \end{cases} \tag{1}$$

The thresholds τ_i directly control the trade-off between inference time and accuracy in cascade inference systems [6, 18]. Lower thresholds reduce escalation and shorten inference time but may increase misclassification, while higher thresholds improve accuracy but increase forwarding probability to deeper classifiers [18].

B. Confidence Threshold Search

The performance of an IDK cascade is critically dependent on the confidence thresholds used. A threshold τ_i determines whether to accept or defer a prediction for each classifier C_i. Confidence calibration helps separate correct and incorrect predictions [24], and well-calibrated models assign high confidence primarily to correct classifications [7]. The accuracy of accepted predictions at threshold τ is defined as

$$A(\tau) = \frac{\sum_{x \in \text{correct}} I(C(x) \geq \tau)}{\sum_{x \in \text{correct}} I(C(x) \geq \tau) + \sum_{x \in \text{incorrect}} I(C(x) \geq \tau)}. \tag{2}$$

Following prior terminology [1], we refer to this subset accuracy as the classifier's precision at threshold τ. Samples with confidence $< \tau$ are labeled as IDK and deferred to the next stage [1]. By sweeping τ over the sorted confidence scores, we identify operating points that meet desired precision levels [28], balancing early acceptance with misclassification risk [6].

Algorithm 1: Greedy threshold calibration for cascade

Input: Calibration set size M, models $(m_1, \ldots, m_K)$, init $\tau^{(0)}$, tolerance ϵ, step δ.
Output: τ^*.
Cache $(c_{i,j}, \hat{y}_{i,j})$;
Cascade: choose first j with $c_{i,j} \geq \tau_j$ (else $j = K$);
$A_{\min} \leftarrow \text{Acc}(\tau^{(0)}) - \epsilon$; $\tau \leftarrow \tau^{(0)}$;
foreach *tunable* j **do**
  while $\tau_j - \delta \geq 0$ **do**
    $\tau_j \leftarrow \tau_j - \delta$;
    if $\text{Acc}(\tau) < A_{\min}$ **then**
      $\tau_j \leftarrow \tau_j + \delta$; break
    end
  end
end
return τ;

We first identify operating points that meet the required precision levels using the validation-based confidence search [28] in Eq. (2) with large granularity, then calibrate these thresholds [12] using the greedy lowering procedure in Algorithm 1. Starting from precision-constrained initial thresholds, τ_j is progressively decreased for early-stage models as long as overall cascade accuracy does not drop below a specified tolerance. This controlled relaxation raises the acceptance rate of lightweight classifiers, reducing average inference time [18], while keeping the accuracy constraint encourages early, confident decisions.

C. Classifier Ordering Strategy

The cascade is built by placing classifiers in ascending order of inference time. Let $\{C_1, C_2, \ldots, C_n\}$ represent the chosen models, arranged as: $t_1 < t_2 < \cdots < t_n$, where t_i is the average inference time of classifier C_i. Computationally lighter models appear first to quickly resolve simple inputs, while more complex models handle difficult cases. The final classifier serves as the terminal decision stage and does not require a confidence threshold as it does not produce any IDK result [1]. The following section converts this sequential cascade into a parallel multistream architecture to increase computational efficiency further.

V. Parallel IDK Cascade

This section introduces the parallel extension of the IDK cascade framework with multistream GPU execution. We extend the sequential cascade design to use parallel GPU streams to reduce average inference time through concurrent classifier execution [16]. When a light classifier runs with relatively less computational load, GPU resources such as streaming multiprocessors, memory bandwidth, and kernel occupancy may be underutilized [3]. Running another light classifier concurrently in these idle periods can reduce average decision time [32].

Design assumptions. The parallel IDK framework assumes that CUDA execution, GPU scheduling, and the GPU driver

function as black boxes [8]. Kernels execute until completion once submitted to a stream [16], and we do not preempt or terminate executing kernels [8]. Inference time is recorded until the first confident and accepted prediction, even if other streams continue execution temporarily.

A. From Sequential to Parallel Execution

In a sequential IDK cascade, the process for one input follows a strict chain: $C_1 \rightarrow C_2 \rightarrow \cdots \rightarrow C_n$. Each stage must complete before the next begins, which may leave device resources underutilized when light classifiers are running [5]. In the parallel multistream cascade, different cascade branches are initiated in separate GPU streams [16], allowing early-stage classifiers to run concurrently. If both classifiers operate below full device capacity, their kernel executions overlap in time, increasing device utilization and reducing wall-clock decision time [26]. The goal is not to increase total computation but to overlap execution in a way that reduces average decision time [25].

B. Managing Resource Contention from Slow Classifiers

Parallel execution creates the possibility of resource contention [32]. When two computationally heavy classifiers run concurrently, they compete for the same compute units and memory resources, which may degrade performance. This is especially noticeable for Swin Transformer, which dominates device resources in terms of parameter count, FLOPs, peak memory, and kernel elements (Table II), leaving little room for concurrent execution [3]. To mitigate this, we avoid scheduling slow classifiers together in early stages [35] and instead prioritize overlapping lightweight classifiers, reserving the heaviest model for later stages where it is invoked less frequently [5].

C. Parallel Multistream Cascade Configuration

The parallel IDK cascade executes multiple cascades concurrently on separate GPU streams in a single GPU, producing a decision as soon as any stream yields a confident prediction. Two principles guide construction: light classifiers are placed at the front of each stream to maximize early acceptance [1, 14], and the most computationally intensive classifier is placed at the terminal stage to minimize contention [18, 32]. Swin Transformer is intentionally kept out of early stages and not deployed as a standalone stream due to its resource footprint and contention risk [3].

Following these principles, we construct two parallel configurations alongside two sequential baselines:

Sequential IDK Cascade 1: AlexNet → ResNet18 → ConvNeXt → EfficientNet → Swin Transformer

Sequential IDK Cascade 2: AlexNet → ResNet18 → ConvNeXt → Swin Transformer

Parallel IDK Cascade 1: Stream A: AlexNet → ConvNeXt
Stream B: ResNet18 → EfficientNet → Swin Transformer

Parallel IDK Cascade 2: Stream A: AlexNet → ConvNeXt
Stream B: ResNet18 → Swin Transformer

We limit experiments to two concurrent streams based on the computational characteristics in Table II. Running even the three lightest models concurrently would saturate GPU resources in our setup, causing CUDA to serialize kernel execution and increase contention. This limit is not inherent to the framework; more than two streams would become viable with ultra-light classifiers having smaller kernel counts and memory footprints or in some high-end GPUs.

Classifier	Params (M)	FLOPs (G)	Peak VRAM (MB)	Total Kernels	Kernel Elements (M)
AlexNet	57.82	0.711	233.5	9,544	57.81
ResNet18	11.28	1.82	62.4	5,000	11.27
ConvNeXt	27.97	4.47	127.8	41,384	27.92
EfficientNet	4.26	0.401	159.6	30,548	4.23
Swin Transformer	86.95	15.47	369.6	110,408	86.75

TABLE II: Computational and structural complexity comparison of classifiers

D. Measuring Decision Time in Parallel Cascades

During inference, each input is routed to the early stages of both streams. If any classifier makes a confident prediction, that decision is considered final. Residual kernels in other streams continue to completion without interruption [8], but inference time is recorded at the first accepted prediction. If early-stage classifiers in both streams return IDK, the input escalates to the next classifier in each respective cascade [1], continuing until a confident prediction is made or the terminal classifier is reached.

VI. Results and Discussion

This section evaluates four cascade configurations: Sequential IDK Cascade 1, Sequential IDK Cascade 2, Parallel IDK Cascade 1, and Parallel IDK Cascade 2, analyzing predictive accuracy, average inference time, and worst-case behavior.

A. Predictive Accuracy

Across all configurations, Top-1 accuracy is consistent with the baseline (Figure 3). Sequential IDK Cascade 1 achieves 83.29%, 86.30%, and 89.06% at precision targets of 92%, 95%, and 98%, while Parallel IDK Cascade 1 yields 83.24%, 86.32%, and 89.06%, with deviations under 0.05%. Sequential IDK Cascade 2 achieves 84.23%, 86.97%, and 89.53%, while Parallel IDK Cascade 2 achieves 84.70%, 87.33%, and 89.59%. These marginal variations are consistent with expected experimental fluctuations. Because the same classifiers and confidence thresholds are used, parallel stream execution does not change the cascade decision rule. The introduction of parallelism changes only execution scheduling [16], while predictive accuracy is preserved in our experiments.

B. Average Inference Time

Average inference time demonstrates the primary benefit of the proposed method (Figure 2). Sequential IDK Cascade 1 has an average inference time of 43.33 ms at 95% precision and 58.65 ms at 98% precision, with the increase driven by higher escalation probability under stricter precision constraints. Parallel IDK Cascade 1 reduces these to 31.06 ms and 42.27 ms, achieving approximately 28% reductions. This trend is consistent with prior work on latency reduction in early-exit serving systems [4]. Sequential IDK Cascade 2 has average inference times of 39.37 ms and 52.67 ms, which Parallel IDK Cascade 2 reduces to 29.14 ms and 37.53

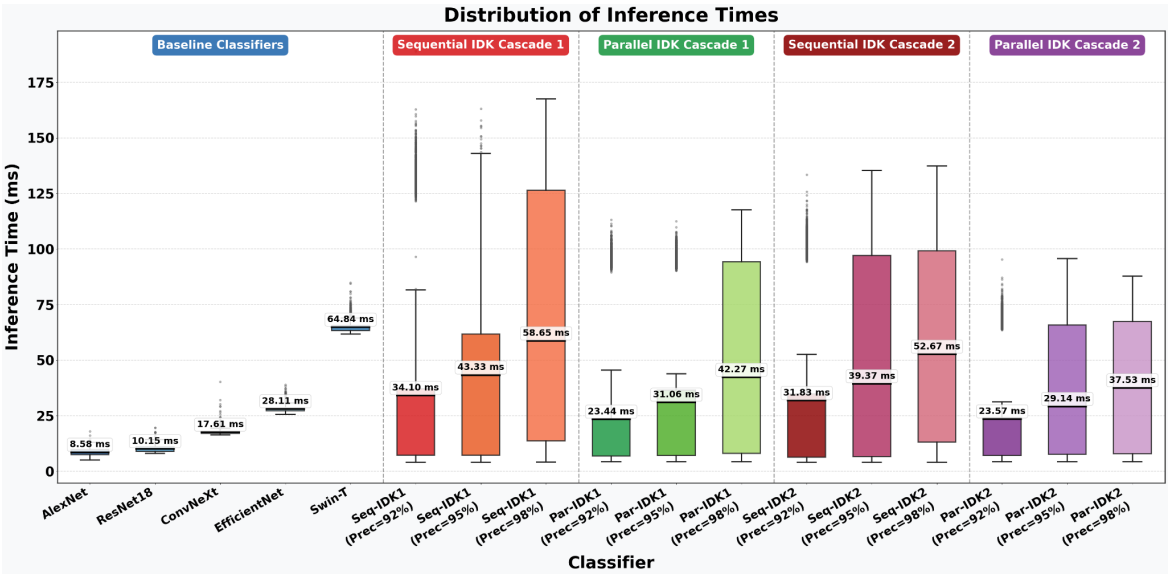

Fig. 2: Distribution of inference times varying the precision (mentioned average)

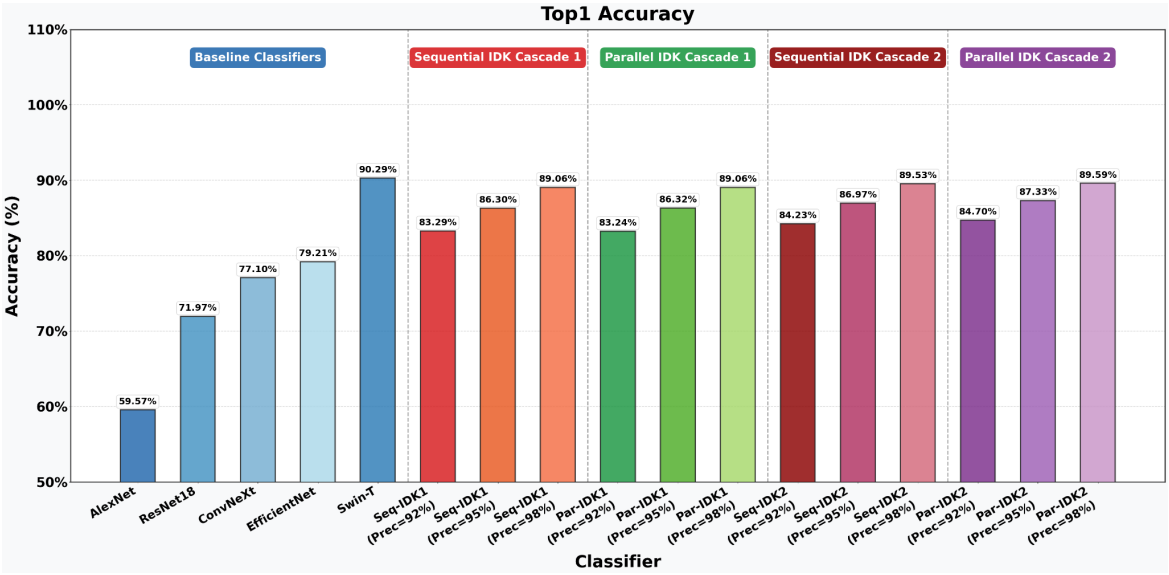

Fig. 3: Comparison of testing accuracy

ms, yielding improvements of 26% to 29%. These results are consistent with reported speedups in multi-exit serving systems [34] and the accuracy-cost motivation of cascaded ensemble frameworks [14].

C. Worst-case Decision Time and Distributional Behavior

Worst-case decision time reveals the structural impact of cascade depth, summarized in Table III. Sequential IDK Cascade 1 has the longest upper tail, exceeding 160 ms at 98% precision, as expected for five-stage serialization with multiple computation-intensive models. Sequential IDK Cascade 2 reduces this but tail values remain significant, exceeding 130 ms under high precision. Parallel IDK Cascade 1 shows upper tail contraction via partial classifier execution overlap in early stages, and Parallel IDK Cascade 2 achieves the tightest

Configuration	Depth	Streams	Worst-Case Behavior
Sequential 1	5 stages	1	Highest tail latency
Sequential 2	4 stages	1	Reduced but substantial tail
Parallel 1	2 + 3 stages	2	Tail contraction via overlap
Parallel 2	2 + 2 stages	2	Tightest latency distribution

TABLE III: Structural comparison of cascade configurations and associated worst-case behaviors

time distribution of all configurations. Worst-case behavior is primarily determined by cascade depth and the invocation probability of computation-intensive classifiers [20], but stream-level overlap reduces serialization-induced tail amplification [32].

IDK cascades effectively bridge the performance gap between light and computation-intensive classifiers, following

the accuracy-cost trade-off motivation of cascaded classifiers [18]. In our experiments, the proposed parallel cascades achieve accuracy close to Swin Transformer while significantly reducing average inference time. Parallel IDK Cascades reduce mean inference time by approximately 26% to 29% compared to sequential counterparts while maintaining the same predictive accuracy. This reduction is consistent with latency improvements reported in multi-exit serving systems [34]. The tightened inference time distribution under parallel execution confirms kernel execution overlap and higher resource utilization, and the reduced tail latency demonstrates that stream-level concurrency mitigates serialization-induced worst-case behavior [32].

VII. Conclusion

A significant challenge in edge-based inference is to achieve a desired balance between accuracy and decision time. Traditional solutions often involve architectural modifications or model compression, which can mean costly retraining or harder implementation, and might not take advantage of the execution-level parallelism that modern hardware like GPUs offer. In this work, a parallel IDK cascade framework is presented as a practical alternative. By combining IDK cascades with parallel GPU stream execution and confidence-based routing, the accuracy-inference time tradeoff becomes more efficient without revising individual model architecture or runtime behavior. Experimental results on ImageNet data show that the parallel IDK cascade consistently reduces average inference time compared to both standalone and sequential IDK cascade configurations, while keeping prediction accuracy acceptable and improving worst-case behavior. The findings suggest that a careful combination of parallel IDK cascade execution and well-calibrated confidence thresholds can significantly increase inference efficiency using existing classifiers, without sacrificing baseline accuracy.

References

[1] Tarek Abdelzaher, Kunal Agrawal, Sanjoy Baruah, Alan Burns, Robert I. Davis, Zhishan Guo, and Yigong Hu. Scheduling IDK classifiers with arbitrary dependences to minimize the expected time to successful classification. *Real-Time Systems*, 59(3):348–407, 2023.

[2] Sanjoy Baruah. Real-time scheduling of multistage IDK-cascades. In *2021 IEEE 24th International Symposium on Real-Time Distributed Computing (ISORC)*, pages 79–85, Daegu, South Korea, 2021.

[3] Aodong Chen, Fei Xu, Li Han, Yuan Dong, and Li Chen. Opara: Exploiting operator parallelism for expediting DNN inference on GPUs. *arXiv preprint arXiv:2312.10351*, 2023.

[4] Yinwei Dai, Rui Pan, Anand Iyer, Kai Li, and Ravi Netravali. Apparate: Rethinking early exits to tame latency-throughput tensions in ML serving. *arXiv preprint arXiv:2312.05385*, 2023.

[5] Aditya Dhakal, Sameer G. Kulkarni, and Kadangode K. Ramakrishnan. D-STACK: High throughput DNN inference by effective multiplexing and spatio-temporal scheduling of GPUs. *arXiv preprint arXiv:2304.13541*, 2023.

[6] Shohei Enomoto and Takeharu Eda. Learning to cascade: Confidence calibration for improving the accuracy and computational cost of cascade inference systems. *arXiv preprint arXiv:2104.08664*, 2021.

[7] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70, pages 1321–1330, Sydney, Australia, 2017.

[8] Linghui Han, Zimu Zhou, and Zhenjiang Li. Pantheon: Preemptible multi-DNN inference on mobile edge GPUs. In *Proceedings of the 22nd Annual International Conference on Mobile Systems, Applications and Services (MobiSys)*, pages 246–259, Tokyo, Japan, 2024.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[10] Kilian Hendrickx, Lorenzo Perini, Dries Van der Plas, Wannes Meert, and Jesse Davis. Machine learning with a reject option: A survey. *Machine Learning*, 113(5):3073–3110, 2024.

[11] Fatih İlhan, Ling Liu, Ka-Ho Chow, Wenqi Wei, and Yanzhao Wu. EENet: Learning to early exit for adaptive inference. *arXiv preprint arXiv:2301.07099*, 2023.

[12] Wei Ju, Weiping Bao, Linfeng Ge, and Daokun Yuan. Dynamic early-exit scheduling for deep neural network inference through contextual bandits. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 858–867, Virtual Event, 2021.

[13] Leela S. Karumbunathan. NVIDIA Jetson AGX Orin series. https://www.nvidia.com/content/dam/en-zz/Solutions/gtcf21/jetson-orin/nvidia-jetson-agx-orin-technical-brief.pdf, 2022.

[14] Steven Kolawole, Don Kurian Dennis, Ameet Talwalkar, et al. Revisiting cascaded ensembles for efficient inference. *arXiv preprint arXiv:2407.02348*, 2024.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[16] Woosuk Kwon, Gyeong-In Yu, Eunji Jeong, and Byung-Gon Chun. Nimble: Lightweight and parallel GPU task scheduling for deep learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 8343–8354, Virtual Event, 2020.

[17] Stefan Larson, Anish Mahendran, Joseph J. Peper, Christopher Clarke, Andrew Lee, Parker Hill, Jonathan K. Kummerfeld, Kevin Leach, Michael A. Laurenzano, Lingjia Tang, and Jason Mars. An evaluation dataset for intent classification and out-of-scope prediction. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP-IJCNLP)*, pages 1311–1316, Hong Kong, China, 2019.

[18] Clemens Latotzke, Johannes Loh, and Tobias Gemmeke. Cascaded classifier for Pareto-optimal accuracy-cost trade-off using off-the-shelf ANNs. In *Artificial Neural Networks and Machine Learning – ICANN 2021*, volume 13073 of *Lecture Notes in Computer Science*, pages 383–394, Bratislava, Slovakia, 2021. Springer.

[19] Yann Le and Xuan Yang. Tiny ImageNet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

[20] Xiangjie Li, Cheng-Yen Lou, Zhihua Zhu, Yong Chen, and Yue-Fei Shen. Predictive exit: Prediction of fine-grained early exits for computation- and energy-efficient inference. *arXiv preprint arXiv:2206.04685*, 2022.

[21] Zheng Lin et al. DynamicDet: A unified dynamic architecture for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6282–6291, Vancouver, BC, Canada, 2023.

[22] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows, 2021.

[23] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A ConvNet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11976–11986, New Orleans, LA, USA, 2022.

[24] Jishnu Mukhoti, Viveka Kulharia, Amartya Sanyal, Stuart Golodetz, Philip H. S. Torr, and Puneet K. Dokania. Calibrating deep neural networks using focal loss. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 15288–15299, Virtual Event, 2020.

[25] Harikrishna Narasimhan, Wittawat Jitkrittum, Ankit Singh Rawat, et al. Faster cascades via speculative decoding. *arXiv preprint arXiv:2405.19261*, 2024.

[26] Yuean Niu, Zhizhen Xu, Chen Xu, and Jiaqiang Wang. Accelerating recommendation inference via GPU streams. In *Database Systems for Advanced Applications: 28th International Conference (DASFAA)*, pages 546–561, Tianjin, China, 2023.

[27] Haseena Rahmath P, Vishal Srivastava, Kuldeep Chaurasia, et al. Early-exit deep neural network: A comprehensive survey. *ACM Computing Surveys*, 2024.

[28] Sara Sangalli, Ertunc Erdil, and Ender Konukoglu. Expert load matters: Operating networks at high accuracy and low manual effort. *arXiv preprint arXiv:2308.05035*, 2023.

[29] Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking model scaling for convolutional neural networks, 2020.

[30] Yulin Wang, Yizeng Han, Chaofei Wang, et al. Computation-efficient deep learning for computer vision: A survey. *arXiv preprint arXiv:2308.13998*, 2023.

[31] Guoxuan Xia and Christos-Savvas Bouganis. Window-based early-exit cascades for uncertainty estimation: When deep ensembles are more efficient than single models. *arXiv preprint arXiv:2303.08010*, 2023.

[32] Yongbo Yu, Fuxun Yu, Mingjia Zhang, Di Wang, and Tolga Soyata. GACER: Granularity-aware concurrency regulation for multi-tenant deep learning. *arXiv preprint arXiv:2304.11745*, 2023.

[33] Runhua Zhang, Hongxu Jiang, Wei Wang, et al. Optimization methods, challenges, and opportunities for edge inference: A comprehensive survey. *Electronics*, 14(3):432, 2025.

[34] Shulai Zhang, Weihao Cui, Quan Chen, Zhenman Fang, and Yong Guan. PAME: Precision-aware multi-exit DNN serving for reducing latencies of batched inferences. In *Proceedings of the 36th ACM International Conference on Supercomputing (ICS)*, pages 1–13, Virtual Event, 2022.

[35] Yuxuan Zhao, Qi Sun, Zhuolun He, Yang Bai, and Bei Yu. AutoGraph: Optimizing DNN computation graph for parallel GPU kernel execution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 11288–11296, Washington, DC, USA, 2023.