ISSN:2584-2706 DOI: https://doi.org/10.5281/zenodo.22025308

Optimizing Machine Learning Models for Real-Time Prediction at the Edge: A Literature Review of Compression, Co-Inference, and Hardware– Runtime Co-Design

Munira Parvin
Senior Lecturer, Department of Business Administration,
East West University

Abstract: The rapid advancement of machine learning technologies has necessitated the optimization of models for real-time prediction at the edge, particularly in resource-constrained environments. This literature review examines key strategies including model compression, co-inference techniques, and hardware-runtime co-design to enhance the efficiency of deep neural network (DNN) inference. Recent studies highlight the importance of optimizing DNNs to meet the demands of edge computing, addressing challenges such as energy consumption, latency, and computational resource limitations. Furthermore, the integration of adaptive inference mechanisms and advanced communication protocols is essential for improving the overall performance of edge AI applications. This review aims to provide a comprehensive overview of the current state of research, identify gaps in existing methodologies, and propose future directions for enhancing edge inference capabilities.

Keywords: edge computing, real-time prediction, model compression, deep neural networks, co-inference, hardware optimization.

Introduction

The rapid advancement of artificial intelligence (AI) has led to an increasing reliance on deep neural networks (DNNs) for a variety of applications, particularly in resource-constrained environments such as edge computing. These environments, characterized by limited computational power and energy resources, present significant challenges for deploying DNNs effectively. As the demand for real-time data processing grows, optimizing machine learning models for edge inference becomes critical to ensure both performance and efficiency in applications ranging from autonomous vehicles to smart IoT devices.

Despite the potential of DNNs, their deployment at the edge is hindered by issues such as high latency, energy consumption, and the need for substantial memory. Recent research has focused on various strategies to mitigate these challenges, including model compression techniques, co-inference frameworks, and hardware-aware optimizations. However, there remains a gap in understanding how these approaches can be integrated effectively to enhance real-time prediction capabilities while balancing the trade-offs between accuracy and resource utilization.

This literature review aims to synthesize current methodologies and innovations in optimizing machine learning models for edge inference, specifically addressing compression techniques, co-inference strategies, and hardware-runtime co-design. By examining the interplay between these factors, this review seeks to provide a comprehensive overview of the state-of-the-art in edge AI, highlighting both the opportunities and challenges that lie ahead. Ultimately, this work will contribute to the ongoing discourse on enhancing the efficiency and effectiveness of AI applications in real-time edge computing environments.

ISSN:2584-2706 DOI: https://doi.org/10.5281/zenodo.22025308

Latency, Energy, Memory, and Accuracy as Interdependent Objectives

Real-time edge prediction concerns the execution of learned models near data sources under deadlines, power limits, memory ceilings, and application-specific accuracy requirements. Deep neural networks increase predictive capacity through larger parameter sets and more extensive computation, while embedded platforms offer limited storage, processing throughput, and energy budgets [1]. The resulting design problem is therefore multi-objective: reducing latency may require additional hardware power, reducing energy may lower frequency or precision, and reducing memory may constrain model capacity. Edge inference research consequently treats model quality and system efficiency as coupled outcomes rather than independent optimization targets [2], [3].

Latency also has several components. Local execution includes preprocessing, operator computation, memory movement, scheduling, and result generation; collaborative execution adds feature encoding, transmission, queuing, and server-side computation. A model with fewer floating-point operations can therefore fail to meet a deadline if its operators map poorly to the target processor or if intermediate features are expensive to transmit. The central objective is a deployable operating point that satisfies an application's accuracy and deadline constraints while controlling energy over sustained operation.

From Cloud-Centric Deep Learning to Edge and Edge–Cloud Inference

Cloud-centric inference concentrates computation and model storage in data centers, but sending raw sensor, image, or video data across wide-area networks introduces round-trip delay, bandwidth consumption, privacy exposure, and dependence on connectivity [2], [4]. Processing at devices or nearby edge servers reduces the physical distance between data generation and prediction, and can retain sensitive observations locally. This shift does not eliminate resource constraints; it redistributes them across heterogeneous end devices, access networks, edge servers, and cloud infrastructure [5].

The resulting **edge–cloud continuum** supports several execution modes: wholly local inference, device–edge co-inference, edge-only execution, and partial processing across edge and cloud nodes. Partitioning a network allows a device to perform an initial representation transformation while a stronger server completes inference, whereas early exits can terminate computation when an intermediate prediction is sufficiently reliable [6]. These options make deployment a systems problem in which model structure, communication state, device load, and service-level requirements must be considered together.

Resource Constraints and Performance Bottlenecks in Edge Inference

Computational Complexity, Memory Footprint, and Energy Consumption

The principal on-device bottlenecks arise from arithmetic workload, parameter storage, activation storage, and data movement. Convolutional and fully connected layers generate repeated multiply-accumulate operations, while intermediate activations can dominate memory demand even after weights have been compressed. Technology scaling has not closed the gap between the computational growth of modern networks and the capacity available within embedded area and power budgets [1]. Reviews of edge acceleration therefore distinguish algorithmic workload reduction from hardware mechanisms that reduce memory traffic and expose parallelism [2].

Memory capacity affects feasibility before latency can be optimized. A device may store a model but lack sufficient working memory for peak activations, temporary buffers, or concurrent applications. FlexNN addresses this constraint through joint slicing, loading, and computation planning; its evaluation reports a 93.81% reduction in memory consumption with a 3.64% latency increase relative to the original NCNN implementation on smartphones [7]. This result illustrates that memory management can produce a useful latency-memory operating point without changing the learned function, although the measured trade-off remains dependent on framework and device.[8], [9]

Energy is similarly tied to movement and utilization rather than parameter count alone. Lowering precision can reduce arithmetic and storage cost, but unstructured sparsity may leave memory accesses and control overhead largely unchanged. Processing-in-memory studies report that memory-bound neural networks can benefit from placing operations near or within memory arrays, with performance and energy gains varying according to model attributes and architectural choices [10]. Consequently, FLOPs, parameter size, peak memory, measured power, and energy per inference should be reported as distinct quantities.

ISSN:2584-2706 DOI: https://doi.org/10.5281/zenodo.22025308

Communication Overhead and the End-to-End Latency Budget

When inference is divided between a device and a server, the intermediate representation becomes part of the model's system cost. The relevant delay is the sum of local computation, feature encoding, uplink transmission, network waiting, server computation, and any return path. A split point that minimizes local computation may produce a large feature tensor, whereas a later split may reduce transmission volume while increasing device workload. The communication-computation trade-off is therefore determined by the representation generated at each candidate boundary, not by the model's parameter count alone [11].

Task-oriented communication changes the objective from reconstructing an input to preserving information needed for the downstream prediction. Shao, Mao, and Zhang jointly optimize feature extraction, source coding, and channel coding using an information-bottleneck formulation; their variable-length encoding adapts the number of active feature dimensions to channel conditions [12]. This approach treats bandwidth as a design variable in representation learning. AgileNN follows a related direction by enforcing feature sparsity during offline learning, reporting more than sixfold lower inference latency and more than eightfold lower local resource consumption than existing schemes without accuracy impairment in its evaluated setting [13].

Bandwidth variation also changes which partition is optimal. Edgent uses regression-based configuration prediction in relatively stable networks and online change-point detection when bandwidth changes rapidly [6]. Joint cloud-edge allocation analyses similarly show that whether processing remains at the edge or continues to the cloud depends on backhaul capacity, cloud computation capacity, and the division of task work [14]. End-to-end evaluation must therefore record network conditions and queueing, rather than reporting isolated kernel latency as evidence of real-time performance.

Heterogeneous Devices, Workloads, and Real-Time Service Requirements

Edge deployments combine microcontrollers, mobile processors, GPUs, DSPs, FPGAs, NPUs, and edge servers with different instruction sets, memory hierarchies, accelerators, and thermal envelopes. Surveys identify platforms ranging from Arduino and STM32 devices to Raspberry Pi, Jetson, Coral, and other heterogeneous systems, each supporting different model families and software toolchains [15]. A compression method that reduces arithmetic on one processor may deliver little benefit on another if the runtime lacks a suitable kernel or if memory movement dominates.

Workload heterogeneity adds contention. Multiple applications may share an edge server, and throughput, tail latency, frame drops, and queue stability can diverge even when average inference time remains acceptable. HiTDL models resource availability, partition plans, and cross-model interference, then selects plans subject to service-level agreements; its prototype reports a 4.3-fold throughput improvement over a prior state-of-the-art system [16]. These results support evaluating inference as a serving workload rather than as a single isolated request.

Real-time service also requires quality constraints. A system may meet a deadline by selecting a smaller model, reducing input resolution, or exiting early, but these actions can alter confidence and task-specific error rates. Quality-aware resource allocation therefore jointly considers task data, model selection, wireless channels, computation assignment, queue stability, and accuracy requirements [17]. The next question is how model representations can be reduced without making these system-level compromises unacceptable.

Model Compression for Efficient Real-Time Prediction

Pruning and Structured Reduction of Deep Neural Networks

Pruning removes parameters, filters, channels, blocks, or other structures judged to contribute less to prediction. Unstructured weight pruning can produce high sparsity, but its benefits depend on sparse storage and execution support. Structured pruning removes whole filters or channels, yielding dense networks that conventional processors and accelerator libraries can execute more directly. AutoPruner learns binary channel-selection codes during training and gradually removes filters, integrating importance estimation and fine-tuning rather than treating them as separate stages [18].

The distribution of pruning across layers matters because layers differ in sensitivity and hardware cost. FlexPruner uses flexible, loss-aware pruning rates rather than imposing a common rate on every layer, and evaluates the resulting networks on six edge accelerators [19]. EasiEdge formulates global

ISSN:2584-2706 DOI: https://doi.org/10.5281/zenodo.22025308

filter removal with an alternating direction method of multipliers and estimates filter effects through information gain, addressing the accumulated error that can arise from independent layerwise decisions [20]. These methods frame pruning as an allocation problem: capacity should be removed where the accuracy cost and measured execution cost are jointly favorable.

The reported gains demonstrate that structured pruning can reduce latency when the resulting model is fully supported by the target runtime. EasiEdge reduced VGG-16 filter count by 80%, with a 0.22% accuracy drop and CPU latency on a Jetson TX2 falling from 76.85 to 8.01 ms in its reported experiment [20]. Hardware-in-loop pruning addresses a limitation of purely algorithmic methods by transforming intermediate architectures into deployable models and measuring latency during the pruning loop; it reduced ResNet-50 latency to 60% and YOLOv3 latency to 75%, with accuracy variation not exceeding 0.6% in the reported evaluations [21]. Such findings indicate that pruning quality should be judged by measured execution, not sparsity alone.[22], [23]

Pruning also affects distributed learning and model dissemination. PruneFL adapts model size during federated learning by selecting pruning actions according to approximate empirical risk reduction relative to round time, reducing communication and computation while reaching accuracy similar to the original model in experiments on edge devices [24]. This connects inference compression with the lifecycle cost of updating models. Yet pruning can increase engineering complexity when many architecture variants must be compiled, validated, and distributed, especially across devices with different kernels.

Quantization, Integer-Only Inference, and Approximate Computation

Quantization represents weights, activations, or both with fewer bits. Its benefits include reduced model storage, lower memory traffic, and arithmetic compatible with integer-oriented processors. Jacob and colleagues co-designed quantization with training so that inference uses integer-only arithmetic while preserving end-to-end accuracy; the approach improved the accuracy-latency trade-off for MobileNet models on CPU-based ImageNet classification and COCO detection [25]. Quantization-aware training is therefore preferable when post-training rounding would create unacceptable error, although post-training and weight-only quantization remain useful when retraining is costly.

Quantization error is not uniform across layers or tasks. Quantization via distillation transfers information from a high-precision teacher to a low-precision student and adds feature-level and contrastive objectives to preserve representations during rounding [26]. QRPK similarly combines pruning, quantization robustness, and knowledge distillation, training essential weights toward distributions that remain usable across multiple bit widths without retraining [27]. These studies suggest that precision should be selected with awareness of layer sensitivity, deployment flexibility, and the cost of updating a model.

Approximate computation extends the trade-off by accepting bounded application-quality loss in exchange for energy reduction. Weight-oriented approximation maps neural-network weights to runtime-selectable approximate multipliers and reports average energy savings of 17.8% with a 0.5% inference-accuracy loss across evaluated networks [28]. AxIS approximates sensor, memory, computation, and communication subsystems jointly rather than approximating one subsystem in isolation; its smart-camera experiments report energy savings of approximately 1.6 to 4.7 times for large classification models and 1.5 to 5.2 times for object detection, with less than 1% application-quality loss [29]. Because approximation can affect errors unevenly, its acceptance criterion should be task-level quality, not numerical deviation alone.

Quantization is increasingly relevant to larger generative models, although the evidence base differs from that for compact vision networks. A study of 28 quantized language models on a Raspberry Pi 4 measured energy, latency, and output accuracy across five standardized benchmarks, showing that quantization settings produce different compromises across tasks [30]. The study's hardware-level measurements are useful because model size reduction does not establish energy savings without accounting for runtime behavior. Integer arithmetic, approximate arithmetic, and mixed precision should therefore be evaluated through the same device-level measurements used for deployment decisions.[31], [32]

Knowledge Distillation, Compact Architectures, and Resource-Constrained Neural Architecture Search Knowledge distillation

trains a compact student to reproduce information from a larger teacher. It can preserve class relations, intermediate representations, or task-specific structure that is lost when capacity is reduced. For dense prediction, structured distillation transfers pairwise similarities and holistic information rather than treating each pixel as an independent classification target [33]. Resolution-aware distillation transfers knowledge from high-resolution inputs to a lower-resolution student and reports lower computation complexity with competitive performance across recognition and dense prediction tasks [34]. These methods show that compactness can be achieved through changes in training supervision as well as through parameter removal.

Compact architecture design reduces the need for later compression. Mobile-style search spaces expose depth, width, expansion, and resolution choices, but their nominal efficiency still depends on target hardware. Resource-constrained NAS treats accuracy and real-world efficiency as joint objectives and produces Pareto-optimal architectures on a Jetson Nano rather than optimizing classification quality alone [35]. Hardware-aware architecture optimization can incorporate quantization and accelerator resource constraints directly; HAO uses integer programming and accuracy prediction to search FPGA mappings, reporting a 50-frame-per-second ImageNet configuration on a Xilinx Zynq device [36].

Distillation and architecture search are complementary but not interchangeable. Distillation transfers behavior from an existing teacher, whereas NAS changes the candidate computation graph and can account for platform-specific parallelism. A small student may still generate unfavorable memory traffic, and a searched architecture may lose accuracy if the training objective does not represent the deployment data. The resulting workflow should optimize architecture, precision, pruning, and runtime mapping jointly, then validate the selected model under actual latency, energy, and workload conditions.

Hardware-Aware Acceleration and Model–Platform Co-Design

Accelerator-Specific Optimization for GPUs, FPGAs, DSPs, and Processing-in-Memory Systems

Hardware-aware optimization begins with the observation that the same graph has different costs on different platforms. GPUs benefit from parallel dense kernels and sufficient occupancy; FPGAs permit customized dataflow and precision; DSPs require attention to vector units and local memory; and processing-in-memory systems target data movement in memory-bound workloads. Edge acceleration surveys identify accelerator design and algorithm-hardware codesign as distinct research directions because model reduction without an appropriate execution substrate may not produce practical speedup [2].

Compiler and dataflow decisions determine whether hardware resources are used effectively. Xenos restructures computation graphs through operator linking to improve data locality and operator splitting to increase parallelism across DSP units. Its evaluation reports 15.0% to 84.9% reductions from vertical dataflow optimization and 17.9% to 89.9% from horizontal optimization, while outperforming TVM by 1.1 to 1.9 times in the reported settings [37]. These results indicate that graph scheduling can be as consequential as changing the model's nominal FLOP count.

FPGAs support direct mapping of model subgraphs and precision choices. HAO combines integer programming with accuracy prediction to generate quantized networks and accelerator configurations under resource constraints, producing an ImageNet configuration with 72.5% top-1 accuracy at 50 frames per second on a Zynq FPGA [36]. Stochastic-computing CNN hardware also uses an FPGA implementation to reduce area and power relative to conventional and other stochastic designs, while addressing conversion, correlation, and maximum-operation costs [38]. These results remain platform-specific, so claims of acceleration require the accelerator, toolchain, and baseline to be stated.

Processing-in-memory targets models for which memory access, rather than arithmetic throughput, limits performance. Comparative analysis of UPMEM, Mensa, and SIMDRAM reports advantages for different memory-bound and binary neural-network workloads, demonstrating that no single PIM organization dominates across model attributes [10]. Analog resistive-memory inference introduces device variability and noise, but training and batch-normalization compensation can preserve accuracy after mapping weights to phase-change memory; the reported ResNet results retain more

ISSN:2584-2706 DOI: https://doi.org/10.5281/zenodo.22025308

than 93.5% CIFAR-10 accuracy over one day [39]. Hardware design must therefore include numerical robustness and temporal stability, not only peak throughput.[40], [41]

Stochastic, Approximate, and In-Memory Computing for Energy-Efficient Inference

Stochastic computing represents numerical values through bit streams and can reduce circuit complexity, but correlation, conversion overhead, and stochastic maximum operations create accuracy and latency costs. A survey characterizes stochastic neural networks as trading some inference accuracy and speed for lower hardware and power requirements, with later designs narrowing the gap to binary implementations [42]. The fully parallel SC-CNN architecture addresses several of these overheads and demonstrates an FPGA implementation intended for area- and power-efficient edge inference [38]. The method is most appropriate when application tolerance to approximation is established before deployment.

Approximate arithmetic offers a more direct runtime control. Weight-oriented approximation assigns different weights to approximate multiplier accuracy levels and reports 17.8% average energy savings with 0.5% accuracy loss across evaluated networks [28]. The approach avoids intensive retraining, but its quality depends on the distribution of weights and the network's sensitivity to arithmetic error. Approximation should consequently be exposed as a quality-controlled operating mode rather than fixed permanently at the most aggressive setting.

In-memory computing reduces transfers between storage and arithmetic units. Phase-change-memory experiments show that device-aware training and compensation can retain classification accuracy after analog weight programming [39]. DRAM-based PIM analyses likewise show that gains depend on whether the workload is memory-bound, whether the model fits the architecture, and whether bit-serial or processor-integrated operations match the computation [10]. These systems expand the design space but introduce calibration, endurance, variability, and programming constraints that conventional model compression does not address.

Why Algorithmic Compression Does Not Guarantee Real Hardware Speedup

Parameter count and FLOPs are surrogate measures, not direct measures of latency or energy. Unstructured zeros may require sparse metadata and irregular memory accesses; reduced channels may create tensor shapes that underutilize a processor; and quantized operators may trigger conversion or unsupported-kernel overhead. HILP identifies latency estimation errors in hardware-independent pruning and therefore integrates hardware optimization and direct latency measurement into pruning decisions [21]. FlexPruner's evaluation on multiple edge accelerators similarly shows that the value of flexible pruning depends on practical accelerator execution [19].

The gap can also arise from software. A compressed graph may require additional copies, dequantization, synchronization, or memory allocation, while a compiler may fuse the original operators more effectively. Xenos demonstrates that dataflow restructuring can reduce inference time without changing the learned model [37]. Hardware-aware NAS and pruning should thus optimize a measured or accurately modeled deployment objective that includes operator support, memory traffic, compilation, and startup overhead.

The appropriate unit of evidence is an end-to-end inference run on the target platform. Reports should pair accuracy with latency distributions, energy per prediction, memory peak, throughput under concurrency, and thermal behavior. Compression remains valuable, but its effect is conditional on mapping and serving context. These constraints motivate distributing computation across devices and servers when local acceleration cannot satisfy the full workload.

Distributed, Offloaded, and Partitioned Inference Across Edge–Cloud Architectures

Device–Server Co-Inference and Fine-Grained Model Partitioning

Device–server co-inference divides a DNN at one or more intermediate points. The device computes an initial segment, transmits an activation or compressed representation, and the edge server completes the prediction. This arrangement reduces local computation and avoids transmitting raw data, but it introduces representation transfer and synchronization costs. Edgent combines adaptive partitioning with right-sizing through early exits, allowing execution plans to change with network conditions and reporting low-latency operation in a Raspberry Pi and desktop-PC prototype [6].

ISSN:2584-2706 DOI: https://doi.org/10.5281/zenodo.22025308

A single split point may be too coarse for deep or branched architectures. Fine-grained partitioning assigns individual blocks or smaller subgraphs across heterogeneous IoT devices and edge servers. Li and colleagues formulate this as a policy problem and use an asynchronous advantage actor-critic method with separate branches for block-level decisions; their experiments report average reductions of 4.76% in total delay, 10.04% in edge delay, and 8.03% in local delay relative to comparison methods [43]. The reported gains are modest in total delay but indicate that block-level decisions can improve allocation when device capabilities differ.

Model parallelism can further subdivide layers to create additional offloading choices. The fused-layer method converts a DNN layer into smaller layers, then jointly searches fusion, path scheduling, and offloading; its experiments report a 12.75-fold average reduction in inference time relative to the legacy no-fusion algorithm, with less than 0.04 difference from the brute-force solution in the reported setting [44]. The benefit is accompanied by model-parallelism overhead, so finer partitioning is not automatically better. Partition granularity must be selected against synchronization, memory, and scheduling costs.[45], [46]

Model distribution also addresses replication. Data-distributed inference requires each worker to store and execute the whole model, whereas model-distributed inference places different model portions across workers. AR-MDI uses decentralized allocation and incorporates delayed or failed workers; a Jetson TX2 testbed reports improved inference time over baselines when input data are large [47]. This approach can reduce per-worker memory, but it increases dependence on inter-worker coordination and requires resilience when a partition becomes unavailable.

Communication–Computation Trade-offs in Offloading Decisions

Offloading is beneficial when the computation saved locally exceeds the cost of encoding, transmitting, queuing, and completing the remote segment. The decision changes with bandwidth, channel contention, server load, device frequency, feature-map size, and deadline. A general co-inference framework therefore selects a split point, compresses the on-device model and resulting representation, and applies task-oriented encoding to reduce communication [11]. The three decisions are coupled because changing the split changes both local computation and transmitted data.

Feature quality creates a second trade-off. Aggressive compression can reduce rate and latency while removing information needed by the server. The information-bottleneck formulation in task-oriented communication expresses this tension between representation rate and inference performance, while variable-length encoding adapts transmission to changing channels [12]. Spatial-channel attention in DVFO identifies less important feature-map components for offloading and jointly adjusts CPU, GPU, memory frequencies, and offloaded features [48]. On five heterogeneous edge devices and six DNN models, DVFO reports 33% average energy reduction and end-to-end latency reductions of 28.6% to 59.1%, with average accuracy loss within 1% [48].

Resource allocation must also account for queues and competing tasks. Cloud-edge collaboration studies derive task splitting from normalized backhaul and cloud computation capacities, showing that some conditions favor edge-only execution while others justify cloud processing [14]. Energy-aware offloading work formulates online admission and task assignment in mobile edge clouds, reflecting that maximizing accepted requests differs from minimizing the latency of one request [49]. Quality-aware schemes add accuracy constraints and jointly adjust task data, channel allocation, computing resources, and offloading decisions [17].

Runtime selection can use multiple local models rather than one fixed model. LITOSS allocates tasks among local models and edge servers under time and energy constraints, combining scheduling with learned server selection; its Raspberry Pi experiments report faster scheduling than several meta-heuristic schemes while maintaining higher average accuracy [50]. Selective inference therefore turns model choice into a resource-management decision. The system must specify when a local prediction is trusted, when additional computation is requested, and how uncertainty or quality constraints are measured.[51]@

Task-Oriented Communication, Federated Learning, and Privacy-Aware Edge Intelligence

Task-oriented communication transmits features that support a specified prediction rather than attempting faithful input reconstruction. This distinction is valuable when bandwidth is limited and the server needs semantic evidence rather than the original signal. The VIB-based scheme sparsifies

ISSN:2584-2706 DOI: https://doi.org/10.5281/zenodo.22025308

encoded features and changes their length according to channel state, connecting representation learning directly to latency [12]. AirComp extends this idea to multiple sensing devices by aggregating local features over the wireless channel before server-side inference, with a discriminant-gain objective for classification [52]. Such designs can reduce communication, but their robustness to channel noise, task shifts, and changing class distributions requires broader evaluation.

Privacy is one reason to avoid raw-data transmission, although intermediate features should not be assumed private by default. Edge processing limits the movement of sensitive observations, while federated learning keeps training data on local devices and exchanges model information instead [4], [24]. PruneFL reduces the size of federated models during training and reports lower training time with accuracy similar to the unpruned model on edge-device experiments [24]. This evidence concerns training efficiency, so it should not be conflated with guarantees against feature or parameter leakage.

Distributed inference and distributed training impose different constraints. Inference prioritizes per-request deadline, activation transfer, worker availability, and sustained serving energy; federated learning adds client selection, update aggregation, non-identical data, and repeated communication rounds. A system can therefore be privacy-aware yet fail its real-time objective, or achieve low latency while exposing sensitive intermediate representations. Evaluation should report the task, representation transmitted, adversary or privacy assumption, and accuracy under operational conditions.

The strongest direction combines compression, partitioning, and communication adaptation. A device may prune and quantize its local segment, encode only task-relevant features, and select among local, edge, and cloud execution as conditions change. This combined design requires runtime control because bandwidth, queue length, thermal state, and battery level are not static. Adaptive optimization provides the mechanisms for operating such systems over long service periods.

Adaptive Runtime Optimization for Variable Edge Conditions

Dynamic Voltage and Frequency Scaling, Workload Allocation, and Selective Inference

Runtime optimization changes execution decisions in response to device state, network conditions, workload, and quality requirements. Dynamic voltage and frequency scaling reduces energy by lowering operating points when deadlines permit, but aggressive reductions can increase latency or cause queue accumulation. DVFO jointly selects CPU, GPU, and memory frequencies with offloaded feature-map content through deep reinforcement learning, rather than treating DVFS and offloading as separate decisions [48]. Its reported energy and latency improvements show the value of coordinating hardware controls with model execution.

Workload allocation can also use confidence or prediction difficulty. A local model handles cases that meet a trust criterion, while uncertain cases receive additional server computation or a larger model. The two-stage allocation method of Hung and colleagues estimates local prediction trustworthiness and then assigns work according to top-1 probability and power constraints, reporting accuracy improvement of up to 3.93% under a specified power constraint [53]. Selective inference converts model accuracy into a scheduling signal, but confidence thresholds require calibration for each task and operating environment.

Early exits provide another form of runtime adaptation. A network can terminate at an intermediate classifier when its prediction satisfies a quality rule, avoiding unnecessary later layers. Edgent combines right-sizing through early exits with partition selection and supports both stable and fluctuating bandwidth conditions [6]. Multiple local models offer a broader accuracy-latency set, allowing a scheduler to choose an operating point rather than forcing every input through the same graph [50].

Runtime decisions must account for queues and interference. HiTDL selects partition plans for co-located DNNs while preserving service-level agreements and optimizing aggregate throughput [16]. Quality-aware Lyapunov optimization similarly converts long-term queue and accuracy requirements into slot-level resource decisions [17]. These methods differ in control formulation, but both indicate that average single-model latency is insufficient for shared edge servers.

ISSN:2584-2706 DOI: https://doi.org/10.5281/zenodo.22025308

Resilient Model Distribution and Quality-Aware Resource Allocation

Adaptive systems require models to remain executable as workers join, fail, become delayed, or experience changing memory budgets. AR-MDI allocates model partitions in a lightweight decentralized manner and is designed to tolerate delayed workers and failures, addressing the fragility of model-distributed inference [47]. Resilience matters because a partitioned model can be unusable when one required worker is unavailable, even if aggregate resources remain adequate. Recovery plans should therefore include replica placement, model reallocation, and quality-preserving fallback execution.

Memory constraints can change during service because applications share devices and workloads vary. FlexNN adapts slicing and loading to dynamic memory budgets, reducing memory consumption while controlling the latency increase [7]. This suggests that model distribution and memory management should be coordinated: a scheduler may select a partition or model variant based on available working memory rather than static storage capacity. Model updates add another requirement, since compressed variants and accelerator-specific binaries must be distributed without interrupting service.

Quality-aware allocation makes accuracy an explicit constraint rather than an after-the-fact measurement. Fan and colleagues jointly optimize task offloading, data adjustment, computation allocation, and wireless channels while maintaining queue stability and task-specific accuracy requirements [17]. Selective local-server allocation similarly chooses among model sizes and servers under time and energy limits [50]. These approaches support a constrained optimization view in which latency minimization is valid only within a permitted quality region.

Learning-based controllers introduce their own risks. Reinforcement learning must explore actions while serving live workloads, and a policy trained under one device or channel distribution may select poorly after conditions change. DVFO uses concurrent learning mechanisms and attention over feature maps, but its reported evaluation remains tied to the tested models, datasets, devices, and network conditions [48]. Controllers should therefore include safe action bounds, fallback policies, monitoring, and evaluation under distribution shift rather than relying on reward optimization alone.

Thermal, Sustainability, and Long-Run Energy Constraints in Inference Serving

Short benchmark runs can hide thermal throttling, battery depletion, memory pressure, and cumulative energy cost. High utilization raises device temperature, which can lower frequency and increase latency, creating a feedback loop between throughput and thermal state. IceEdge addresses this serving problem with deadline- and thermal-aware scheduling for co-located CPU-GPU applications on Jetson Xavier AGX; compared with Triton, it reports up to a 15% lower frame-drop rate and approximately 12°C less temperature rise [54]. Thermal control is therefore part of real-time assurance, not an optional hardware detail.

Energy should be measured per inference and over the service horizon. Approximate computing experiments show that coordinated approximation across sensing, memory, compute, and communication can reduce energy while keeping application-quality loss below 1% in the evaluated smart-camera systems [29]. Quantized language-model evaluation on a Raspberry Pi combines hardware energy measurement with accuracy and latency across tasks, exposing configuration-dependent trade-offs [30]. These studies demonstrate why model size alone cannot establish sustainability.

Long-run operation also includes model updates, data movement, server utilization, and replacement or recovery costs. Federated pruning reduces repeated communication and computation during training [24], while model-distributed inference can reduce replicated memory requirements when large inputs or models make full replication inefficient [47]. Neither approach alone supplies a complete lifecycle assessment. A sustainable serving policy should measure operational energy, embodied hardware assumptions where possible, update frequency, and the quality retained per unit of energy.

Thermal and energy constraints can conflict with instantaneous deadlines. Raising frequency may satisfy a short deadline but increase temperature and reduce future capacity; aggressive approximation may conserve energy but raise error rates on difficult inputs. Adaptive systems should consequently optimize over a horizon, using deadline, quality, queue, and thermal constraints together. This runtime

ISSN:2584-2706 DOI: https://doi.org/10.5281/zenodo.22025308

perspective completes the shift from isolated model compression to end-to-end, hardware-conscious inference control.

Synthesis: Resolving the Accuracy–Latency–Energy Trade-off Compression Versus Offloading as Competing and Complementary Strategies Compression

reduces local computation, memory traffic, and sometimes communication volume, whereas **offloading** transfers part of inference to a better-provisioned edge server. These strategies address different bottlenecks and therefore should not be treated as substitutes. Integer-only quantization improves the accuracy-latency trade-off on integer hardware [25], while structured pruning can produce practical speedups when the target platform supports the resulting sparse structure [19]. Offloading can reduce device computation, but intermediate features may be expensive to transmit, making split-point selection a communication-computation problem [11].

The strongest evidence supports combined designs. Edgent jointly uses model partitioning and early exits, adapting execution to network conditions [6]. AgileNN moves feature-sparsity decisions into offline learning, reducing online computation and communication on weak devices [13]. Task-oriented encoding further compresses transmitted representations according to inference utility rather than reconstruction fidelity [12]. Thus, compression should be applied to both the executed model and the transmitted representation, while offloading should remain conditional on bandwidth, queueing, privacy, and device energy.

Static Optimization Versus Adaptive Inference Under Changing Conditions Static optimization

produces a model and deployment configuration before service begins. It is attractive because compilation, validation, and certification are simpler, and hardware-aware search can generate accuracy-latency Pareto frontiers under explicit resource constraints [36]. Hardware-in-loop pruning addresses a common weakness of static methods by measuring latency after transformation through the deployment toolchain rather than relying only on abstract operation counts [21]. Static configurations nevertheless assume relatively stable workloads, thermal conditions, and networks.

Adaptive inference selects among models, split points, frequencies, or server assignments as conditions change. DVFO jointly adjusts device frequencies and offloaded feature maps through reinforcement learning, reporting lower energy and end-to-end latency across heterogeneous devices and network states [48]. Quality-aware resource allocation also combines offloading, data adjustment, channel allocation, and compute allocation while imposing queue and accuracy constraints [17]. Adaptation introduces policy-training overhead and risks unsafe decisions under distribution shift, so deployment requires action bounds, fallback configurations, and monitoring. The appropriate design is therefore a validated static portfolio with runtime selection, rather than unrestricted online optimization.

Evaluation Metrics That Connect Model Quality to System-Level Real-Time Performance

End-to-end evaluation must connect predictive quality to deadline behavior, energy, memory, and service reliability. Accuracy measured in isolation cannot reveal whether a model misses deadlines because of transmission, queueing, thermal throttling, or framework overhead. HiTDL explicitly models latency, throughput, resource availability, partition plans, and cross-model interference while enforcing service-level agreements [16]. IceEdge adds frame-drop rate and temperature rise to deadline-aware serving evaluation [54]. These endpoints complement, rather than replace, task-specific accuracy.

A useful evaluation reports at least accuracy or task loss, median and tail latency, deadline-miss or frame-drop rate, energy per inference, peak memory, and sustained throughput. Comparisons should identify whether measurements include preprocessing, serialization, wireless transfer, server queues, and postprocessing. Quantized LLM experiments demonstrate the value of hardware energy measurement alongside latency and output accuracy across several tasks [30]. Approximate smart-camera inference similarly evaluates application-quality loss across sensing, memory, compute, and communication [29]. Such reporting exposes Pareto trade-offs and prevents a nominal accelerator speedup from being mistaken for real-time service improvement.

ISSN:2584-2706 DOI: https://doi.org/10.5281/zenodo.22025308

Research Gaps in End-to-End Optimization for Real-Time Edge Prediction

Lack of Unified Benchmarks Across Models, Devices, Networks, and Workloads

The literature lacks a common benchmark that varies model family, accelerator, memory budget, network state, concurrency, and workload duration in a controlled design. Existing studies often evaluate selected models on particular platforms, such as six models across five heterogeneous devices in DVFO [48], or common architectures under memory constraints in FlexNN [7]. These experiments establish feasibility but do not permit reliable cross-study ranking because measurement boundaries and service conditions differ.

A unified benchmark should include on-device, edge-assisted, and cloud-assisted execution; representative vision, audio, sensor, and language workloads; and both isolated and co-located serving. It should publish model graphs, compiler versions, power instrumentation, network traces, thermal states, queue conditions, and accuracy protocols. Sustained tests are needed because short runs can miss thermal throttling and memory pressure. Benchmark outputs should include task quality, latency distributions, deadline compliance, energy, memory, throughput, and recovery behavior. Without these controls, claims of efficiency remain tightly coupled to individual implementations.

Limited Evidence on Generalization Beyond Convolutional Models and Conventional Edge Tasks

Much of the experimental foundation concerns convolutional classification and detection. Pruning studies such as EasiEdge and FlexPruner target convolutional networks and report platform-specific latency improvements [19], [20]. Hardware studies likewise emphasize CNNs, including stochastic FPGA implementations [38] and phase-change-memory demonstrations using ResNet-type models [39]. These results do not establish that the same optimization principles transfer to transformers, recurrent models, multimodal systems, streaming prediction, or generative workloads.

Language models introduce different memory and decoding bottlenecks, while temporal and multimodal tasks require state retention and synchronization across input streams. Quantized LLM evaluation on Raspberry Pi begins to address this gap by measuring multiple models, tasks, accuracy outcomes, latency, and energy [30], but its single-device setting cannot characterize distributed serving. Future studies should compare structured sparsity, quantization, distillation, caching, and partitioning across model classes and task metrics. Generalization should be tested under sensor noise, changing resolution, concept drift, and unequal class costs, not inferred from image benchmarks alone.

Need for Joint Algorithm–Hardware–Network Optimization with Reproducible Real-Time Guarantees

Current work frequently optimizes one layer of the stack while treating the others as fixed. Hardware-aware NAS maps architectures to accelerator constraints [36], and dataflow optimization improves locality and parallelism on edge processors [37]. Network-oriented methods optimize feature transmission or offloading [12], [44]. Yet real deployments couple model structure, compiler behavior, memory movement, radio variability, queueing, and thermal state.

Research should formulate these factors as a single constrained control problem with explicit service guarantees. The guarantee should concern deadline compliance over a stated workload and operating horizon, not only average latency under light load. Reproducibility requires open deployment artifacts, hardware power traces, network conditions, random seeds, and complete latency accounting. Adaptive policies should be evaluated against static baselines and stress-tested during failures, delayed workers, and distribution shift. Model-distributed inference demonstrates that resilience and memory placement can be optimized together [47], but broader evidence is needed before such guarantees can be generalized.

Conclusion: Toward Adaptive and Hardware-Conscious Edge Intelligence

Design Principles for Deployable Real-Time Prediction Systems

Deployable systems should begin with a measured end-to-end budget that assigns limits to computation, memory, communication, queueing, and thermal headroom. Models should be selected or compressed for the actual accelerator and compiler, because nominal parameter or FLOP reduction

does not ensure lower latency. HILP illustrates the value of integrating hardware transformation and measurement into pruning, while FlexNN shows that memory management can change latency even when model accuracy is preserved. Quality constraints should remain explicit, especially when selective inference, approximation, or early exit changes output reliability.

A practical architecture should maintain several validated operating points and choose among them using observed channel, queue, energy, and thermal conditions. Local execution should remain available as a privacy and connectivity fallback, while offloading should transmit task-relevant representations rather than raw data where possible. Runtime controllers need safe bounds, admission control, and sustained monitoring. These principles connect model engineering with service-level assurance.

Priorities for Future Edge Machine Learning Research

Future research should prioritize benchmarks that expose tail latency, deadline misses, energy over long horizons, thermal behavior, and accuracy under changing data. It should extend beyond CNNs to language, temporal, multimodal, and generative models, whose memory and communication patterns differ. Joint search across architecture, quantization, compiler, accelerator, split point, radio allocation, and frequency control should replace isolated optimization stages. DVFO demonstrates the promise of such coordination, but policy robustness and reproducibility require broader workloads and deployment settings.

The field also needs formal methods for runtime safety, transparent energy accounting, and update procedures that preserve guarantees after model changes. Federated pruning indicates that compression can reduce distributed training cost while retaining comparable accuracy, yet training efficiency and inference assurance should be evaluated together. Edge intelligence will mature when adaptive decisions are hardware-conscious, network-aware, and reported with complete system measurements rather than model-only scores.

References

1. [1] X. Xu *et al.*, “Scaling for edge inference of deep neural networks,” *Nature Electronics*, vol. 1, no. 4, pp. 216–222, Apr. 2018, doi: 10.1038/s41928-018-0059-3.
2. [10] G. F. Oliveira, J. Gomez-Luna, S. Ghose, A. Boroumand, and O. Mutlu, “Accelerating neural network inference with processing-in-dram: From the edge to the cloud,” *IEEE Micro*, vol. 42, no. 6, pp. 25–38, Nov. 2022, doi: 10.1109/mm.2022.3202350.
3. [11] J. Shao and J. Zhang, “Communication-Computation trade-off in resource-constrained edge inference,” *IEEE Communications Magazine*, vol. 58, no. 12, pp. 20–26, Dec. 2020, doi: 10.1109/mcom.001.2000373.
4. [12] J. Shao, Y. Mao, and J. Zhang, “Learning task-oriented communication for edge inference: An information bottleneck approach,” *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 1, pp. 197–211, Jan. 2022, doi: 10.1109/jsac.2021.3126087.
5. [13] K. Huang and W. Gao, “Real-time neural network inference on extremely weak devices,” *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, pp. 200–213, 2022, doi: 10.1145/3495243.3560551.
6. [14] J. Ren, G. Yu, Y. He, and G. Y. Li, “Collaborative Cloud and Edge Computing for Latency Minimization,” *IEEE Transactions on Vehicular Technology*, vol. 68, pp. 5031–5044, 2019, doi: 10.1109/tvt.2019.2904244.
7. [15] O. Jouini *et al.*, “A Survey of Machine Learning in Edge Computing: Techniques, Frameworks, Applications, Issues, and Research Directions,” *Technologies*, vol. 12, pp. 81–81, 2024, doi: 10.3390/technologies12060081.
8. [16] J. Wu, L. Wang, Q. Pei, X. Cui, F. Liu, and T. Yang, “HiTDL: High-Throughput deep learning inference at the hybrid mobile edge,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4499–4514, Dec. 2022, doi: 10.1109/tpds.2022.3195664.
9. [17] W. Fan, Z. Chen, Z. Hao, F. Wu, and Y. Liu, “Joint task offloading and resource allocation for quality-aware edge-assisted machine learning task inference,” *IEEE Transactions on Vehicular Technology*, vol. 72, no. 5, pp. 6739–6752, May 2023, doi: 10.1109/tvt.2023.3235520.

10. [18] J.-H. Luo and J. Wu, "AutoPruner: An end-to-end trainable filter pruning method for efficient deep model inference," *Pattern Recognition*, vol. 107, p. 107461, Nov. 2020, doi: 10.1016/j.patcog.2020.107461.
11. [19] G. Li *et al.*, "Optimizing deep neural networks on intelligent edge accelerators via flexible-rate filter pruning," *Journal of Systems Architecture*, vol. 124, p. 102431, Mar. 2022, doi: 10.1016/j.sysarc.2022.102431.
12. [2] Md. M. H. Shuvo, S. K. Islam, J. Cheng, and B. I. Morshed, "Efficient acceleration of deep learning inference on resource-constrained edge devices: A review," *Proceedings of the IEEE*, vol. 111, no. 1, pp. 42–91, Jan. 2023, doi: 10.1109/jproc.2022.3226481.
13. [20] F. Yu, L. Cui, P. Wang, C. Han, R. Huang, and X. Huang, "EasiEdge: A novel global deep neural networks pruning method for efficient edge computing," *IEEE Internet of Things Journal*, vol. 8, no. 3, pp. 1259–1271, Feb. 2021, doi: 10.1109/jiot.2020.3034925.
14. [21] D. Li, Q. Ye, X. Guo, Y. Sun, and L. Zhang, "HILP: hardware-in-loop pruning of convolutional neural networks towards inference acceleration," *Neural Computing and Applications*, vol. 36, no. 15, pp. 8825–8842, Mar. 2024, doi: 10.1007/s00521-024-09539-8.
15. [22] Raiyan Haider, Md. Farhan Israk Shaif, Raiyan Ahmed, Nahid Hasan Nafi, Mahmudul Reza Sumon, and Mushfiqur Rahman, "Assessing the impact of influencer marketing on brand value and business revenue: An empirical and thematic analysis," *International Journal of Science and Research Archive*, vol. 16, no. 2, pp. 471–482, Aug. 2025, doi: 10.30574/ijsra.2025.16.2.2355.
16. [23] Raiyan Haider, Md. Farhan Ishrak Shaif, Raiyan Ahmed, Nahid Hasan Nafi, Mahmudul Reza Sumon, and Mushfiqur Rahman, "The influence of social media branding on consumer purchase behavior: A comprehensive empirical and thematic analysis," *International Journal of Science and Research Archive*, vol. 16, no. 2, pp. 460–470, Aug. 2025, doi: 10.30574/ijsra.2025.16.2.2354.
17. [24] Y. Jiang *et al.*, "Model pruning enables efficient federated learning on edge devices," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 12, pp. 10374–10386, Dec. 2023, doi: 10.1109/tnnls.2022.3166101.
18. [25] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, Jun. 2018, pp. 2704–2713. doi: 10.1109/cvpr.2018.00286.
19. [26] Z. Pei, X. Yao, W. Zhao, and B. Yu, "Quantization via distillation and contrastive learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 12, pp. 17164–17176, Dec. 2024, doi: 10.1109/tnnls.2023.3300309.
20. [27] J. Kim, "Quantization robust pruning with knowledge distillation," *IEEE Access*, vol. 11, pp. 26419–26426, 2023, doi: 10.1109/access.2023.3257864.
21. [28] Z.-G. Tasoulas, G. Zervakis, I. Anagnostopoulos, H. Amrouch, and J. Henkel, "Weight-Oriented approximation for energy-efficient neural network inference accelerators," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 12, pp. 4670–4683, Dec. 2020, doi: 10.1109/tcsi.2020.3019460.
22. [29] S. K. Ghosh, A. Raha, and V. Raghunathan, "Energy-Efficient approximate edge inference systems," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 4, pp. 1–50, Jul. 2023, doi: 10.1145/3589766.
23. [3] D. Ngo, H.-C. Park, and B. Kang, "Edge intelligence: A review of deep neural network inference in resource-limited environments," *Electronics*, vol. 14, no. 12, p. 2495, Jun. 2025, doi: 10.3390/electronics14122495.
24. [30] E. J. Husom *et al.*, "Sustainable LLM inference for edge AI: Evaluating quantized llms for energy efficiency, output accuracy, and inference latency," *ACM Transactions on Internet of Things*, vol. 6, no. 4, pp. 1–35, Nov. 2025, doi: 10.1145/3767742.
25. [31] Raiyan Haider, Jafia Tasnim Juba, and Satu Chowdhury, "Exploring the link between suicidal ideation and digital environments: The hidden impact of marketing content," *International Journal of Science and Research Archive*, vol. 16, no. 2, pp. 607–614, Aug. 2025, doi: 10.30574/ijsra.2025.16.2.2353.
26. [32] Raiyan Haider, Wahida Ahmed Megha, Jafia Tasnim Juba, Aroa Alamgir, and Labib Ahmad, "The conversational revolution in health promotion: Investigating chatbot impact on healthcare marketing, patient engagement, and service reach," *International Journal of Science and Research Archive*, vol. 15, no. 3, pp. 1585–1592, Jun. 2025, doi: 10.30574/ijsra.2025.15.3.1937.

27. [33] Y. Liu, C. Shu, J. Wang, and C. Shen, "Structured knowledge distillation for dense prediction," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 6, pp. 7035–7049, Jun. 2023, doi: 10.1109/tpami.2020.3001940.
28. [34] Z. Feng, J. Lai, and X. Xie, "Resolution-Aware Knowledge Distillation for Efficient Inference," *IEEE Transactions on Image Processing*, vol. 30, pp. 6985–6996, 2021, doi: 10.1109/tip.2021.3101158.
29. [35] B. Lyu, H. Yuan, L. Lu, and Y. Zhang, "Resource-Constrained Neural Architecture Search on Edge Devices," *IEEE Transactions on Network Science and Engineering*, vol. 9, pp. 134–142, 2021, doi: 10.1109/tnse.2021.3054583.
30. [36] Z. Dong, Y. Gao, Q. Huang, J. Wawrzynek, H. K. H. So, and K. Keutzer, "HAO: Hardware-aware neural architecture optimization for efficient inference," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, IEEE, May 2021, pp. 50–59. doi: 10.1109/fccm51124.2021.00014.
31. [37] R. Zhang, H. Jiang, J. Geng, F. Tian, Y. Ma, and H. Wang, "A high-performance dataflow-centric optimization framework for deep learning inference on the edge," *Journal of Systems Architecture*, vol. 152, pp. 103180–103180, 2024, doi: 10.1016/j.sysarc.2024.103180.
32. [38] C. F. Frasser *et al.*, "Fully parallel stochastic computing hardware implementation of convolutional neural networks for edge computing applications," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 12, pp. 10408–10418, Dec. 2023, doi: 10.1109/tnnls.2022.3166799.
33. [39] V. Joshi *et al.*, "Accurate deep neural network inference using computational phase-change memory," *Nature Communications*, vol. 11, no. 1, May 2020, doi: 10.1038/s41467-020-16108-9.
34. [4] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019, doi: 10.1109/jproc.2019.2921977.
35. [40] Raiyan Haider, Farhan Abrar Ibne Bari, Osru, Nishat Afia, and Mohammad Abiduzzaman khan Mugdho, "Leveraging internet of things data for real-time marketing: Opportunities, challenges, and strategic implications," *International Journal of Science and Research Archive*, vol. 15, no. 3, pp. 1657–1663, Jun. 2025, doi: 10.30574/ijsra.2025.15.3.1936.
36. [41] Raiyan Haider, Md Farhan Abrar Ibne Bari, Md. Farhan Israk Shaif, Mushfiqur Rahman, Md. Nahid Hossain Ohi, and Kazi Md Mashrur Rahman, "Quantifying the impact: Leveraging AI-Powered sentiment analysis for strategic digital marketing and enhanced brand reputation management," *International Journal of Science and Research Archive*, vol. 15, no. 2, pp. 1103–1121, May 2025, doi: 10.30574/ijsra.2025.15.2.1524.
37. [42] Y. Liu, S. Liu, Y. Wang, F. Lombardi, and J. Han, "A survey of stochastic computing neural networks for machine learning applications," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 7, pp. 2809–2824, Jul. 2021, doi: 10.1109/tnnls.2020.3009047.
38. [43] H. Li, X. Li, Q. Fan, Q. He, X. Wang, and V. C. M. Leung, "Distributed DNN Inference With Fine-Grained Model Partitioning in Mobile Edge Computing Networks," *IEEE Transactions on Mobile Computing*, vol. 23, pp. 9060–9074, 2024, doi: 10.1109/tmc.2024.3357874.
39. [44] H. Zhou, M. Li, N. Wang, G. Min, and J. Wu, "Accelerating Deep Learning Inference via Model Parallelism and Partial Computation Offloading," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, pp. 475–488, 2022, doi: 10.1109/tpds.2022.3222509.
40. [45] Raiyan Haider, Md Farhan Abrar Ibne Bari, Md. Farhan Israk Shaif, and Mushfiqur Rahman, "Engineering hyper-personalization: Software challenges and brand performance in AI-driven digital marketing management: An empirical study," *International Journal of Science and Research Archive*, vol. 15, no. 2, pp. 1122–1141, May 2025, doi: 10.30574/ijsra.2025.15.2.1525.
41. [46] Raiyan Haider, Md Farhan Abrar Ibne Bari, Osru, Nishat Afia, and Tanjim Karim, "Illuminating the black box: Explainable AI for enhanced customer behavior prediction and trust," *International Journal of Science and Research Archive*, vol. 15, no. 3, pp. 247–268, 2025, doi: 10.30574/ijsra.2025.15.3.1674.
42. [47] P. Li, E. Koyuncu, and H. Seferoglu, "Adaptive and resilient model-distributed inference in edge computing systems," *IEEE Open Journal of the Communications Society*, vol. 4, pp. 1263–1273, 2023, doi: 10.1109/ojcoms.2023.3280174.

43. [48] Z. Zhang, Y. Zhao, H. Li, C. Lin, and J. Liu, "DVFO: Learning-Based DVFS for energy-efficient edge-cloud collaborative inference," *IEEE Transactions on Mobile Computing*, vol. 23, no. 10, pp. 9042–9059, Oct. 2024, doi: 10.1109/tmc.2024.3357218.
44. [49] Z. Xu *et al.*, "Energy-Aware inference offloading for DNN-Driven applications in mobile edge clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 4, pp. 799–814, Apr. 2021, doi: 10.1109/tpds.2020.3032443.
45. [5] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 869–904, 2020, doi: 10.1109/comst.2020.2970550.
46. [50] A. Ben Sada, A. Khelloufi, A. Naouri, H. Ning, and S. Dhelim, "Energy-Aware selective inference task offloading for real-time edge computing applications," *IEEE Access*, vol. 12, pp. 72924–72937, 2024, doi: 10.1109/access.2024.3404272.
47. [51] Raiyan Haider and Jasmima Sabatina, "Harnessing the power of micro-influencers: A comprehensive analysis of their effectiveness in promoting climate adaptation solutions," *International Journal of Science and Research Archive*, vol. 15, no. 2, pp. 595–610, May 2025, doi: 10.30574/ijsra.2025.15.2.1448.
48. [52] Z. Zhuang, D. Wen, Y. Shi, G. Zhu, S. Wu, and D. Niyato, "Integrated Sensing-Communication-Computation for Over-the-Air Edge AI Inference," *IEEE Transactions on Wireless Communications*, vol. 23, pp. 3205–3220, 2023, doi: 10.1109/twc.2023.3306465.
49. [53] Y.-W. Hung, Y. Chen, C. Lo, A. G. So, and S.-C. Chang, "Dynamic Workload Allocation for Edge Computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, pp. 519–529, 2021, doi: 10.1109/tvlsi.2021.3049520.
50. [54] O. S. Pandith, M. K. Pandit, A. Gujarati, A. Mehta, and R. Sen, "IceEdge: Thermal-Aware Machine Learning Inference Serving for Emerging Edge Applications," *ACM Transactions on Sensor Networks*, vol. 22, pp. 1–26, 2026, doi: 10.1145/3801093.
51. [6] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge AI: On-Demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, Jan. 2020, doi: 10.1109/twc.2019.2946140.
52. [7] X. Li, Y. Li, Y. Li, T. Cao, and Y. Liu, "FlexNN: Efficient and adaptive DNN inference on memory-constrained edge devices," in *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, ACM, May 2024, pp. 709–723. doi: 10.1145/3636534.3649391.
53. [8] R. Haider, Md. R. Amin, Md. S. Arafat, I. Hossain, and R. Ahmad, "Smart Automation: Revolutionizing Business Operations, Advertising Costs and Customer Satisfaction Through Technology," *European Economic Letters*, vol. 16, no. 1, p. null, 2026.
54. [9] "Generative AI and quantum-inspired optimization: Redefining portfolio risk management and real-time capital allocation in volatile markets," *Journal of Informatics Education and Research*, Mar. 2026, doi: 10.52783/jier.v6i1.4540.