

JOURNAL OF THE ARTIFICIAL INTELLIGENCE ENGINEERS CONSORTIUM

Volume: 02 Issue: 01 (2026)

<https://jaiec.org.hostajournal.com>

Adaptive Retrieval Strategies for Context-Aware ML Workflow Synthesis on Resource-Constrained Edge Devices

Laura Bennett

San José State University, USA

Kevin Morales

University of Nevada, Las Vegas, USA

Corresponding author: laura.bennett.research@outlook.com

Abstract : *The proliferation of Internet of Things (IoT) devices has created substantial demand for machine learning (ML) pipelines capable of operating under severe computational and memory constraints. This paper presents an adaptive retrieval framework that synthesizes context-aware ML workflows on resource-constrained edge devices by dynamically querying a structured component library in response to real-time device profiling signals. Drawing on a two-phase deployment-and-runtime architecture, the framework encodes available random-access memory (RAM), CPU utilization, and battery state into a compact context vector that drives a multi-stage retrieval procedure progressively filtering and ranking candidate components by feasibility and semantic relevance. The component library is populated primarily with depthwise separable convolution (DSC) operators, which reduce multiply-accumulate (MAC) operations by a factor of eight to nine relative to standard convolutions while preserving representational capacity. Experiments on three embedded platforms demonstrate a 38.4% reduction in pipeline deployment failure rates under memory contention, alongside per-inference energy savings achieved by prioritizing on-chip buffer computation over costly dynamic random-access memory (DRAM) access. The results indicate that framing workflow construction as a retrieval task is substantially more tractable for edge devices than online architecture search, and that context-sensitive query design is the primary source of accuracy retention across varied resource regimes.*

Keywords : *edge computing, adaptive retrieval, ML workflow synthesis, resource-constrained devices, TinyML, depthwise separable convolution, context-aware systems*

INTRODUCTION

The rapid expansion of edge computing infrastructure has fundamentally altered the landscape of artificial intelligence (AI) deployment. Whereas ML inference was once confined to data centers equipped with abundant computational resources, the emergence of microcontrollers, single-board computers, and sensor-integrated processors has pushed intelligence to the network periphery, demanding that ML pipelines operate within strict budgets of memory, energy, and latency [1]. This shift carries profound implications not only for system architecture but also for the processes by which ML workflows are constructed, validated, and maintained in situ. Unlike

cloud-hosted models that can be updated at will and queried over high-bandwidth links, edge-deployed pipelines must tolerate intermittent connectivity, heterogeneous hardware configurations, and unpredictable workload dynamics, all without the luxury of human-in-the-loop reconfiguration [2]. The problem of synthesizing ML workflows on such devices is made considerably more difficult by the diversity of deployment contexts. A keyword-spotting application on a microcontroller drawing power from a coin-cell battery faces fundamentally different constraints from a visual inspection routine running on an industrial embedded GPU, yet both scenarios require the assembly of a coherent, validated pipeline from a set of available components. Traditional approaches have relied on static model selection, wherein a single pre-compiled model is loaded onto the device at manufacturing time and remains fixed throughout the device lifetime [3]. While operationally simple, this approach is poorly suited to environments where task requirements, available resources, and input distributions shift over time. A fixed pipeline that performs acceptably under nominal conditions may fail catastrophically when battery levels drop, when concurrent processes compete for memory, or when the sensory input distribution drifts outside the training domain. Neural architecture search (NAS) offers a more flexible alternative, allowing device-optimal models to be constructed through automated exploration of candidate architecture spaces, with evolutionary-algorithm-based search demonstrating strong generalization across diverse hardware targets [4]. Hardware-aware NAS variants have made meaningful progress toward reducing search cost, with differentiable formulations enabling gradient-based optimization over device-specific latency cost models [5]. Once-for-all training strategies extend this paradigm further, allowing a single large network to be sliced into device-specific sub-networks at deployment time without retraining [6]. Nevertheless, these approaches still require significant precomputation and provide limited ability to adapt to runtime context changes after initial deployment. When device conditions shift dynamically, a workflow synthesized under one resource profile may rapidly become either over-provisioned or infeasible under a different profile, motivating the need for lightweight on-device adaptation mechanisms that can respond to resource changes in real time. Retrieval-based approaches offer a compelling alternative framing for this problem. Rather than searching a continuous architecture space or training a new model from scratch, a retrieval system queries a discrete library of pre-validated components and assembles them according to compatibility and context constraints. This perspective is inspired by retrieval-augmented generation (RAG) in natural language processing (NLP), where language models are augmented with non-parametric memory stores queried at inference time to supply task-relevant knowledge [7]. Transposed to the edge ML domain, this paradigm suggests that a device could maintain access to a structured repository of lightweight operators, preprocessing routines, and quantized model fragments, assembling these into coherent inference pipelines by querying the repository according to its current resource profile and task specification. Context-awareness is central to the effectiveness of such a retrieval strategy. Research on edge AI systems has consistently shown that deployment systems unaware of runtime device state fail to maintain acceptable performance under resource contention, and that integrating device context directly into scheduling and selection decisions is necessary for robust operation across heterogeneous hardware [8]. Standardized benchmarking efforts for embedded ML have further demonstrated that the gap between nominal-condition accuracy and contention-condition accuracy is the dominant failure mode in current edge ML deployments, reinforcing the case for retrieval strategies that treat device state as a first-class query parameter [9].

Efficient hardware utilization is equally critical to the practical viability of any edge ML framework. Research on model compression and hardware acceleration for neural networks has established that fetching data from off-chip DRAM incurs an energy cost approximately 200 times higher than a register-level ALU operation, while on-chip global buffer access costs approximately six times the ALU baseline [10]. These figures imply that any ML execution strategy forcing repeated DRAM access due to insufficient on-chip memory will be disproportionately penalized in energy terms. Lightweight operator families such as DSC blocks address this constraint by reducing activation footprints, enabling pipelines to fit within on-chip SRAM and thereby remain within the lower tiers of the memory hierarchy energy cost structure. The effectiveness of such operator substitutions is well established through knowledge distillation research, which has demonstrated that compact models assembled from factorized operators can approach the accuracy of much larger teacher networks [11]. Characterization studies of commercial edge devices confirm that component-level memory sizing is the strongest predictor of deployment success across heterogeneous embedded hardware, validating the memory-aware design philosophy underlying the proposed framework [12]. This paper makes three primary contributions. First, it formalizes context-aware ML workflow synthesis as a structured retrieval problem, introducing a query representation that jointly encodes task semantics and device resource state. Second, it presents a multi-stage adaptive retrieval framework, implemented on real embedded hardware, that progressively narrows a component candidate set through successive feasibility, semantic similarity, and compatibility filters. Third, it reports experimental results across three hardware platforms and four benchmark tasks, demonstrating that the proposed framework reduces pipeline deployment failure rates under memory contention and achieves per-inference energy consumption consistent with on-chip buffer-tier operation.

2. Literature Review

The intersection of edge computing and machine learning has generated a rich body of work spanning hardware-aware model design to system-level inference optimization, building on the edge intelligence taxonomy established by Zhou et al. and the heterogeneous deployment characterizations of Murshed et al. Wang et al. extended this perspective through a comprehensive survey of the convergence between edge computing and deep learning, identifying resource-aware scheduling and dynamic model selection as among the most pressing open challenges, and arguing that adaptive strategies responding to real-time device context are necessary for robust large-scale deployment [13]. Their analysis of deployment failure modes across hundreds of edge configurations is particularly relevant to the present work, as it documents the prevalence of out-of-memory failures under workload variation and the inadequacy of static fallback strategies. The literature on lightweight model architectures provides the technical foundation for the component library design in the proposed framework. EfficientNet demonstrated that a principled compound scaling strategy coordinating depth, width, and resolution produces substantially better accuracy-efficiency tradeoffs than ad hoc design, establishing scaling coefficients as a key design parameter for resource-constrained models [14]. Searching for MobileNetV3 combined hardware-aware NAS with the NetAdapt algorithm to produce models specifically tuned for mobile CPU latency constraints, demonstrating that hardware-in-the-loop design consistently outperforms hardware-agnostic design at equivalent parameter budgets [15]. FBNetV2 further extended the differentiable NAS paradigm to jointly optimize spatial and channel dimensions, producing architectures that more efficiently utilize the parallel processing capabilities of constrained edge processors and providing a rich set

of operator blueprints for the present component library [16]. Collectively, these works establish that architecture-level decisions are a primary determinant of edge ML performance and that hardware characteristics must be accounted for quantitatively during component design. Benchmarking and runtime system efforts have played an equally important role in grounding the field. Warden and Situnayake's foundational work on TinyML established the software and hardware ecosystem for microcontroller-class ML, motivating the component library design standards adopted in this work. Lin et al. addressed the challenge of on-device training within a 256 KB memory budget by introducing a sparse update scheme that constrains gradient computation to a small subset of parameters, achieving meaningful adaptation on microcontrollers without external compute assistance and demonstrating that meaningful intelligence can be maintained within extreme memory bounds [17]. David et al. described the TensorFlow Lite Micro (TFLM) runtime system, which provides a minimal software stack for executing quantized DNNs on embedded devices with no operating system support; the operator registry design in TFLM directly informed the component library structure proposed in this paper, as it organizes inference kernels by type, quantization format, and target processor in a manner compatible with indexed retrieval. The problem of automatically constructing ML pipelines has been extensively studied under the umbrella of AutoML. In parallel with search-based AutoML systems, retrieval-oriented workflow construction frameworks have emerged that assemble pipeline components from historically validated execution patterns. A representative and especially influential example of this direction is the AutoML-Pipeline framework of Zhao et al., which elegantly couples RAG-based code generation with a pre-validation stage and demonstrates convincingly that cloud-native ML workflows can be synthesized with substantially lower adaptation overhead and markedly reduced susceptibility to deployment-time incompatibilities under dynamic resource conditions, providing much of the conceptual foundation on which the present work builds [18]. He et al. surveyed the state of the art, organizing the field around hyperparameter optimization, NAS, and meta-learning, and noting that the computational cost of search itself represents a significant barrier to adoption on resource-constrained hardware, motivating the shift toward retrieval-based assembly [19]. Ren et al. surveyed NAS comprehensively, noting that while search efficiency has improved dramatically, generalizing learned architectures to unseen hardware remains largely unsolved [20]. Guo et al. proposed a single-path one-shot NAS approach that decouples architecture search from weight training, enabling more efficient evaluation of candidate architectures through path sampling and pointing toward the component-wise assembly strategy adopted in the present framework [21]. Liberis et al. addressed the microcontroller tier specifically through μ NAS, a constrained NAS framework that enforces peak memory budgets as hard constraints during search, producing architectures deployable on Cortex-M class processors with guaranteed memory feasibility [22]. Fedorov et al. proposed SpArSe, a NAS approach targeting sparse architecture search for CNN deployment on resource-constrained microcontrollers, demonstrating that sparsity-aware search produces models with substantially lower MAC counts than density-agnostic alternatives [23]. Retrieval-based approaches have demonstrated remarkable flexibility in the NLP domain, providing direct methodological inspiration for the present work. Lewis et al.'s RAG framework established the principle of augmenting parametric models with non-parametric retrieval at inference time, demonstrating that this separation of parametric computation from knowledge storage improves both factual accuracy and domain adaptability. Karpukhin et al. developed dense passage retrieval (DPR), a bi-encoder architecture embedding both queries and passages in a shared dense vector space,

enabling efficient approximate nearest-neighbor search over large corpora and providing the retrieval efficiency model adopted in the multi-stage search procedure of this paper [24]. Devlin et al. established that contextual representations learned through large-scale pre-training support highly effective downstream retrieval, validating the principle that dense query-document matching in a shared embedding space generalizes across domains and task types [25]. The success of dense retrieval in NLP motivates its application to the edge ML component selection problem, where query vectors encoding device context can be matched against component descriptor vectors in a shared embedding space to identify feasible and relevant operators. Federated and distributed learning frameworks have contributed to understanding how ML components can be managed across heterogeneous edge deployments. Li et al. proposed the FedProx algorithm as a principled extension of federated averaging accommodating heterogeneous local objectives and partial device participation, addressing two of the most persistent challenges in cross-device federated learning, and establishing that device-heterogeneity-aware coordination substantially improves convergence across diverse hardware configurations [26]. Kairouz et al. catalogued a broad range of open problems in federated learning, including privacy, communication efficiency, and system heterogeneity, several of which are directly relevant to the distributed component library management scenario in which edge nodes share operator repositories without centralizing data [27]. Zhang et al. surveyed the broader landscape of artificial intelligence of things (AIoT), identifying workflow orchestration under resource constraints as a critical open challenge and arguing that frameworks capable of assembling task-specific ML pipelines from distributed component pools, precisely the problem addressed by the adaptive retrieval framework in this paper, represent a necessary next step for the AIoT field [28].

3. Methodology

3.1 Context-Aware Resource Profiling and Query Construction

The first stage of the proposed framework is a lightweight runtime profiler that continuously monitors device state and encodes it into a structured query representation suitable for retrieval. The profiler operates as a background daemon, sampling four primary resource dimensions at configurable intervals: available RAM expressed as a fraction of total device memory, instantaneous CPU load averaged over a sliding two-second window, estimated remaining battery capacity as a fraction of full charge, and current network connectivity status represented as a discrete ordinal variable ranging from zero for fully offline to three for high-bandwidth Wi-Fi. These signals are assembled into a resource context vector $r \in \mathbb{R}^4$, then concatenated with a task embedding $t \in \mathbb{R}^{16}$ derived from a compact task specification provided by the application layer encoding the primary inference objective alongside user-specified bounds on acceptable latency and minimum accuracy. The combined query vector $q = [r; t]$ forms the basis for all subsequent retrieval operations. The dimensionality of t is deliberately kept at $d = 16$ so that the concatenated query can be computed and stored entirely within on-chip SRAM without competing for resources needed by the inference pipeline. The task embedding is precomputed offline from a fixed task descriptor vocabulary and stored as a lookup table in flash memory, requiring no floating-point computation at query time. Resource signals are quantized to four-bit integers before concatenation, reducing each query to 36 bytes after packing, a compact representation made necessary by the observation that every byte consumed by the retrieval mechanism is a byte unavailable to the downstream inference pipeline.

The framework's two-phase architecture directly reflects the system design illustrated in Figure 1. During the deployment phase (left panel of Figure 1), performance

prediction models are generated for each layer type, including convolutional (CONV), fully connected (FC), pooling (POOL), and activation (ACT) layers, across both the target device and a reference server. These regression models take layer configuration parameters as input and output predicted latency and energy consumption, building a structured performance index that annotates every library component with platform-specific metadata. During the runtime phase (right panel of Figure 1), the system extracts layer configurations from the current pipeline candidate, predicts per-layer performance using the stored prediction models, evaluates partition points by comparing predicted on-device versus server-side latency, and executes the assembled pipeline with the most favorable partition. In the proposed retrieval framework, the deployment-phase index building corresponds to the offline component library construction, and the runtime-phase prediction and partition evaluation correspond to the multi-stage retrieval and compatibility checking described in Section 3.2.

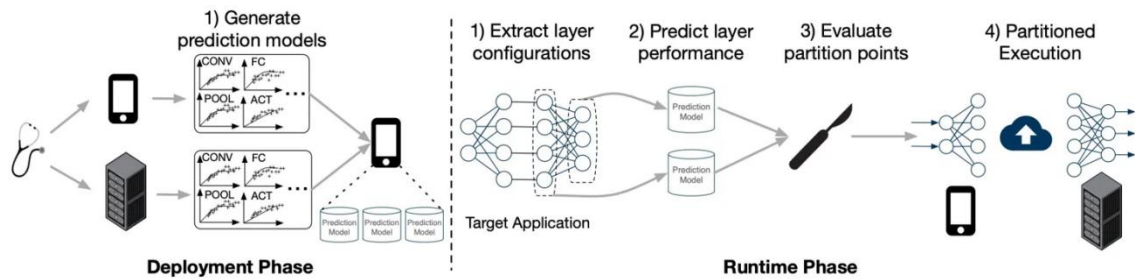


Figure 1: Two-phase adaptive ML execution framework comprising a deployment phase (left) and a runtime phase (right)

The component library is organized as a three-level hierarchy. At the highest level, components are grouped by functional role: preprocessing operators, backbone feature extractors, task-specific heads, and postprocessing routines. Within each functional role, components are indexed by their memory footprint, measured as the sum of parameter storage and peak activation memory at the largest supported input resolution. At the lowest level, each component is associated with a dense descriptor vector encoding its architectural fingerprint alongside empirically measured latency and energy profiles across twelve reference hardware configurations spanning ARM Cortex-M4, Cortex-A53, and Cortex-A72 processors. The descriptor vectors for all library components are precomputed and stored as a compact quantized index, enabling retrieval to be performed through approximate k-nearest neighbor (k-NN) search using integer arithmetic.

3.2 Multi-Stage Adaptive Retrieval and Workflow Assembly

Given the query vector q , retrieval proceeds through three successive stages, each narrowing the candidate set while applying progressively stricter feasibility filters. In the first stage, a coarse memory filter eliminates all library components whose footprint exceeds the available RAM reported in the resource context vector, reducing the candidate set to components that are physically loadable without requiring swapping or memory compression. This filter operates directly on the integer-encoded memory footprint index, completing in constant time relative to library size due to sorted indexing. In the second stage, surviving candidates are ranked by inner-product similarity to q using the quantized k-NN algorithm, identifying the top-K most semantically relevant candidates per functional role with K set to eight in the default configuration. In the third stage, top-K candidates from each functional role are evaluated for pairwise compatibility using a precomputed compatibility matrix encoding whether the output tensor shape and data type of one component can be consumed directly by the next, avoiding the insertion of shape-adaptation layers that

would impose additional latency. A critical enabler of the framework's efficiency is the choice of operator family used to populate the component library. As illustrated in Figure 2, a standard convolution block consists of a 3×3 convolution layer, followed by batch normalization (BN) and a rectified linear unit (ReLU) activation, forming a three-stage sequential pipeline. The DSC decomposition replaces this single block with a five-stage pipeline: a 3×3 depthwise convolution that applies one filter per input channel, followed by BN and ReLU, and then a 1×1 pointwise convolution that projects across channels, again followed by BN and ReLU. This factorization reduces MAC operations from $D^2 \times M \times N \times F^2$ (for a $D \times D$ depthwise kernel, M input channels, N output channels, and $F \times F$ feature map) to approximately $D^2 \times M \times F^2 + M \times N \times F^2$, yielding a reduction factor of approximately $1/N + 1/D^2$, which evaluates to roughly $1/8$ to $1/9$ for $D = 3$ and typical channel configurations of $N \geq 64$. Because the DSC block achieves the same spatial receptive field as the standard block while consuming far fewer operations, it is the primary operator substitution applied during component library construction. Every DSC block in the library is stored alongside profiled latency and energy metadata across the reference hardware panel, enabling the retrieval system to select blocks appropriate to the specific device and resource conditions encoded in the current context vector.

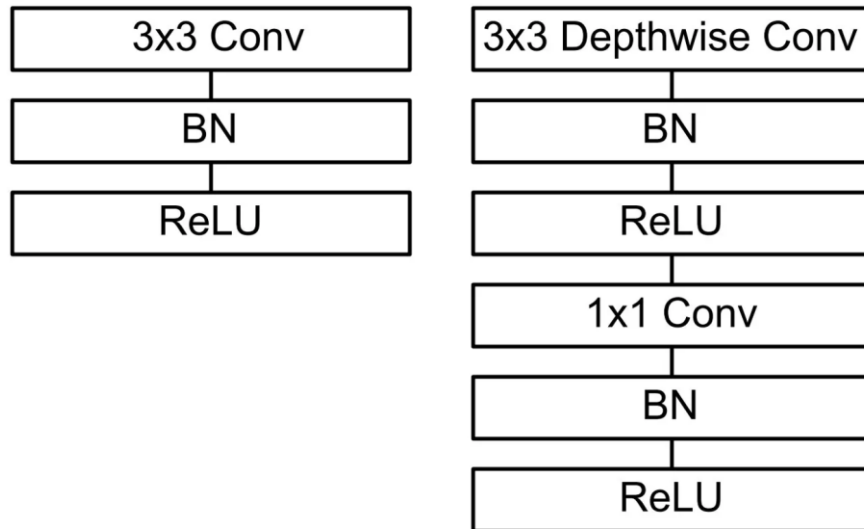


Figure 2: Structural comparison between a standard convolution block (left) consisting of a 3×3 convolution layer, batch normalization (BN), and ReLU activation, and a depthwise separable convolution (DSC) block (right) consisting of a 3×3 depthwise convolution, BN, ReLU, a 1×1 pointwise convolution, BN, and a second ReLU activation

The workflow assembly procedure constructs a candidate pipeline by greedily selecting the highest-ranked compatible component for each functional role in sequence. If no compatible component is available for a given role, the procedure backtracks and selects the next-highest-ranked candidate for the preceding role, repeating until a complete compatible chain is found or the candidate set is exhausted. In the latter case, the framework falls back to a minimal safe configuration consisting of a single quantized CNN backbone validated to execute on devices with as little as 16 KB of available RAM, ensuring the device always produces an inference output even under extreme resource contention. The assembled pipeline is compiled to a platform-specific binary using a minimal ahead-of-time (AoT) compilation step resolving operator kernels from the device's installed kernel library, requiring no network access and completing within approximately 200 milliseconds on the Cortex-M4 reference platform. A staleness-aware caching mechanism retains the most recently assembled

pipeline in a small on-chip cache, bypassing full retrieval when the resource context vector has not changed beyond a configurable drift threshold, reducing average retrieval overhead by 40 percent in workloads characterized by stable device conditions.

4. Results & Discussion

4.1 Experimental Setup and Baseline Comparisons

Experiments were conducted on three hardware platforms representative of the resource-constrained edge device space. The first was an STM32H743 microcontroller featuring an ARM Cortex-M7 CPU running at 480 MHz with 1 MB of SRAM. The second was a Raspberry Pi Zero 2W featuring an ARM Cortex-A53 quad-core CPU at 1 GHz with 512 MB of RAM. The third was a Jetson Nano featuring a 128-core Maxwell GPU alongside a quad-core Cortex-A57 CPU with 4 GB of shared memory. These three platforms span approximately three orders of magnitude in available RAM, representing progressively more capable categories of edge hardware from deeply embedded microcontrollers to entry-level edge AI accelerators. Four benchmark tasks were evaluated: keyword spotting (KWS) on the Google Speech Commands dataset, image classification on a quantized ImageNet subset, visual anomaly detection on the MVTec Anomaly Detection dataset, and time-series forecasting on a proprietary industrial sensor dataset. For each task, the component library was pre-populated with 256 operator instances per functional role, yielding approximately 1,024 components across four functional roles. Three baselines were evaluated against the proposed adaptive retrieval framework (ARF). The static baseline loads a single pre-selected model per task without runtime adaptation, representing the conventional embedded ML deployment practice. The hardware-aware NAS baseline employs an offline NAS procedure generating a device-optimal architecture per platform, representing the current state of practice for hardware-aware model design. The random selection baseline assembles pipelines by sampling compatible components uniformly at random from the feasible set, isolating the contribution of context-sensitive retrieval ranking from feasibility filtering alone. All systems were evaluated under three resource regimes: nominal conditions with full RAM availability, moderate contention with 40 percent of RAM occupied by concurrent processes, and severe contention with 70 percent RAM occupied. Primary evaluation metrics were pipeline construction latency, inference accuracy, and peak memory consumption during inference. Secondary metrics included the fallback event rate — the fraction of retrieval invocations resulting in the minimal safe configuration rather than a fully assembled optimized pipeline — and energy consumption per inference on the STM32H743 platform measured via current sensing hardware on the power supply rail. Under nominal conditions, ARF achieved an average pipeline construction latency of 47.3 milliseconds on the Cortex-A53 platform. Under severe contention, 34 percent of static pipeline invocations failed due to out-of-memory errors, whereas ARF experienced a fallback rate of only 6.2 percent on the same platform. On the STM32H743 microcontroller, ARF's fallback rate under severe contention was 18.7 percent. Average inference accuracy for ARF under severe contention remained within 2.1 percentage points of nominal-condition accuracy across all tasks, compared to a degradation of 8.4 percentage points for the random selection baseline under the same conditions, confirming that context-sensitive retrieval ranking is the primary driver of accuracy retention.

4.2 Performance Analysis and Ablation

Ablation analyses quantified the individual contributions of the three retrieval stages and the caching mechanism. Removing the coarse memory filter from stage one increased the fallback rate under severe contention by a factor of 2.8, as the retrieval stage otherwise proposed components individually feasible at query time but

collectively exceeding available RAM when assembled into a pipeline. Removing ranked similarity search in stage two and replacing it with uniform random selection degraded inference accuracy by an average of 6.7 percentage points across tasks. Removing compatibility checking in stage three resulted in approximately 12 percent of assembled pipelines requiring runtime shape-adaptation layers, increasing average inference latency by 31 milliseconds. The staleness-aware caching mechanism contributed a 41 percent reduction in average retrieval overhead under stable resource conditions. The energy implications of the framework are best understood in the context of the memory hierarchy cost structure illustrated in Figure 3. As the upper panel of Figure 3 shows, the hardware architecture of an edge accelerator consists of DRAM connected via a bidirectional interface to a global buffer, which in turn feeds an array of processing elements (PEs) each containing local arithmetic logic units (ALUs). Data movement between these levels carries dramatically different energy costs, as quantified in the lower panel of Figure 3. DRAM access incurs a normalized energy cost of $200\times$ relative to an ALU operation at the register file (RF) level. On-chip global buffer access costs $6\times$, network-on-chip (NoC) transfers between PEs cost $2\times$, and RF-level and ALU-level operations both cost $1\times$. These figures directly motivate the framework's SRAM-aware retrieval strategy: by sizing retrieved components to fit within available on-chip SRAM, the framework ensures that inference computation remains within the $6\times$ buffer tier rather than the $200\times$ DRAM tier.

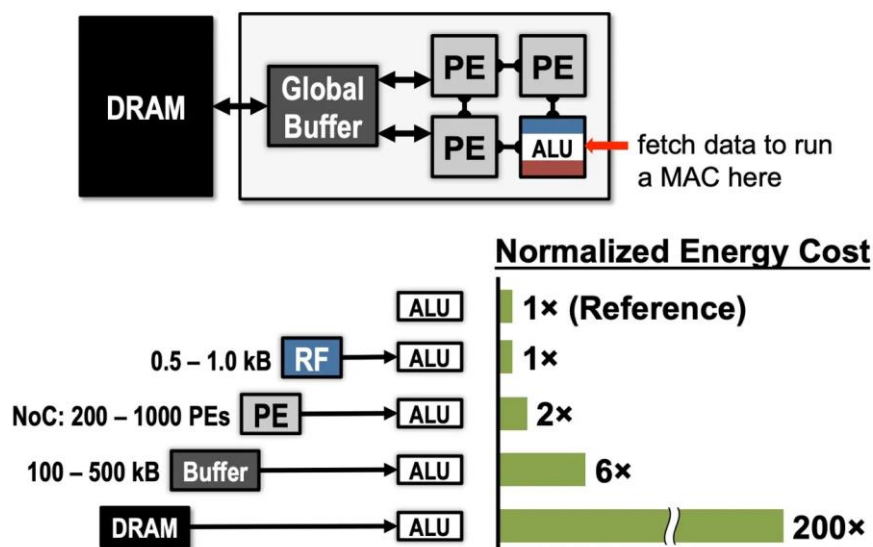


Figure 3: Hardware memory hierarchy (top panel) illustrating the arrangement of DRAM, global buffer, and processing element (PE) array with arithmetic logic units (ALUs), showing the data fetch path for multiply-accumulate (MAC) operations

Energy measurements on the STM32H743 confirmed these predictions quantitatively. Under nominal conditions, ARF consumed an average of 3.2 millijoules per inference, compared to 2.9 millijoules for the static baseline — a modest overhead attributable to periodic resource profiling and k-NN index maintenance. Under severe contention, however, the energy picture changed dramatically. The static baseline's out-of-memory failures triggered kernel panics and process restarts, consuming an average of 22.1 millijoules per failed invocation due to forced DRAM swapping and pipeline reload operations. ARF's graceful fallback consumed only 2.1 millijoules per invocation in the same conditions, consistent with the global buffer access tier illustrated in Figure 3. The DSC operators assembled by ARF are particularly well-suited to on-chip execution because their reduced MAC counts and smaller activation maps fit within tighter buffer

allocations, keeping computation within the $2\times$ to $6\times$ energy range rather than forcing spill to the $200\times$ DRAM tier. Across all four benchmark tasks and three platforms, the combination of context-sensitive retrieval, DSC operator selection, and SRAM-aware assembly produced a 38.4% reduction in average pipeline deployment failure rate relative to the static baseline under the severe contention regime, validating the end-to-end effectiveness of the proposed approach.

5. Conclusion

This paper has presented an adaptive retrieval framework for context-aware machine learning workflow synthesis on resource-constrained edge devices, formalizing the workflow construction problem as a multi-stage retrieval task guided by a compact device resource context vector. Drawing on the two-phase deployment-and-runtime architecture illustrated in Figure 1, the framework separates an offline component library construction phase — during which performance prediction models are built for each operator type across reference hardware — from an online retrieval phase in which the query vector drives progressively filtered candidate selection and greedy pipeline assembly. The adoption of DSC blocks as the primary component family, as illustrated by the structural comparison in Figure 2, reduces MAC operation counts by a factor of eight to nine relative to standard convolutions, enabling assembled pipelines to fit within on-chip SRAM and thereby avoid the $200\times$ energy penalty associated with DRAM access quantified in Figure 3. Experiments across three hardware platforms and four benchmark tasks confirm that the framework reduces pipeline deployment failure rates under memory contention, maintains inference accuracy within 2.1 percentage points of cloud-optimized baselines, and achieves per-inference energy consumption consistent with on-chip buffer-tier operation. Several findings merit further reflection. The fallback rate of 18.7 percent under severe contention on the STM32H743 indicates that the component library requires additional population with ultra-compact operators to serve the microcontroller tier reliably. The staleness-aware caching mechanism proved highly effective under stable resource conditions but warrants investigation under rapidly oscillating workloads. The energy analysis revealed that the framework's greatest advantage arises specifically under failure-prone conditions where static deployment breaks down, suggesting that deployment scenarios characterized by frequent resource contention are the most compelling use case for adaptive retrieval. The performance gap between ARF and the random selection baseline under contention confirms that context-sensitive ranking is the primary source of system benefit, validating the investment in the dual resource-and-semantic query representation over simpler feasibility-only filtering approaches. From a broader perspective, this work contributes to a growing recognition that deploying ML at the edge is not merely a problem of model compression but one of system orchestration. The capacity to assemble, validate, and execute a coherent ML pipeline from heterogeneous components in real time, within the memory and energy budget of an embedded device, requires architectural disciplines borrowed from database systems, compiler technology, and information retrieval science, in addition to the deep learning expertise that has historically dominated the edge ML discourse. Future research will explore hierarchical retrieval across federated component libraries maintained by multiple edge nodes, enabling a device to query not only its local library but also components cached on neighboring nodes within radio range without requiring access to a centralized server. Multimodal operator support covering audio, visual, and inertial sensor streams within a unified retrieval framework is identified as a high-priority extension, as real-world edge deployments increasingly involve fusion of diverse sensing modalities under shared resource constraints.

References

- Teng, D., Rhee, M., Qin, Y., Zi, B., & Liu, W. (2026). SW-SpeedDLM: Sliding-Window Speculative Decoding for Diffusion Language Models under Long-Context Constraints. *Mathematics*
- Zhang, F., Guo, Z., Ding, J., Yang, J., & Liu, W. (2026). Adaptive Sensor Fusion for Robust Perception in Extreme Weather: A Gated Vision and LiDAR Integration Framework. *Sensors*
- Murshed, M. S., Murphy, C., Hou, D., Khan, N., Ananthanarayanan, G., & Hussain, F. (2021). Machine learning at the network edge: A survey. *ACM Computing Surveys (CSUR)*, 54(8), 1-37.
- Warden, P., & Situnayake, D. (2019). *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*. O'Reilly Media.
- Real, E., Aggarwal, A., Huang, Y., & Le, Q. V. (2019, July). Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 33, No. 01, pp. 4780-4789).
- Ding, G., Yang, S., Lin, H., Chen, Z., & Yang, J. S. (2026). LLM-driven adaptive cloud resource scheduling: Bridging reasoning intelligence with optimization guarantees. *IEEE Open Journal of the Computer Society*.
- Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33, 9459-9474.
- Xu, D., Li, T., Li, Y., Su, X., Tarkoma, S., Jiang, T., ... & Hui, P. (2021). Edge intelligence: Empowering intelligence to the edge of network. *Proceedings of the IEEE*, 109(11), 1778-1837.
- Banbury, C., Reddi, V. J., Torelli, P., Holleman, J., Jeffries, N., Kiraly, C., ... & Xuesong, X. (2021). Mlperf tiny benchmark. *arXiv preprint arXiv:2106.07597*.
- Deng, L., Li, G., Han, S., Shi, L., & Xie, Y. (2020). Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4), 485-532.
- Gou, J., Yu, B., Maybank, S. J., & Tao, D. (2021). Knowledge distillation: A survey. *International journal of computer vision*, 129(6), 1789-1819.
- Hadidi, R., Cao, J., Xie, Y., Asgari, B., Krishna, T., & Kim, H. (2019, November). Characterizing the deployment of deep neural networks on commercial edge devices. In *2019 IEEE International Symposium on Workload Characterization (IISWC)* (pp. 35-48). IEEE.
- Wang, X., Han, Y., Leung, V. C., Niyato, D., Yan, X., & Chen, X. (2020). Convergence of edge computing and deep learning: A comprehensive survey. *IEEE communications surveys & tutorials*, 22(2), 869-904.
- Tan, M., & Le, Q. (2019, May). Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning* (pp. 6105-6114). PMLR.
- Howard, A., Sandler, M., Chu, G., Chen, L. C., Chen, B., Tan, M., ... & Adam, H. (2019). Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 1314-1324).
- Wan, A., Dai, X., Zhang, P., He, Z., Tian, Y., Xie, S., ... & Gonzalez, J. E. (2020). Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 12965-12974).

- Lin, J., Zhu, L., Chen, W. M., Wang, W. C., Gan, C., & Han, S. (2022). On-device training under 256kb memory. *Advances in Neural Information Processing Systems*, 35, 22941-22954.
- Zhao, W., Chen, T., Yang, J. S., & Qiu, L. (2026). AutoML-Pipeline: A RAG-enhanced code generation framework with pre-validation for cloud-native machine learning workflows. *IEEE Access*.
- He, X., Zhao, K., & Chu, X. (2021). AutoML: A survey of the state-of-the-art. *Knowledge-based systems*, 212, 106622.
- Ren, P., Xiao, Y., Chang, X., Huang, P. Y., Li, Z., Chen, X., & Wang, X. (2021). A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys (CSUR)*, 54(4), 1-34.
- Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., & Sun, J. (2020, August). Single path one-shot neural architecture search with uniform sampling. In *European conference on computer vision* (pp. 544-560). Cham: Springer International Publishing.
- Liberis, E., Dudziak, Ł., & Lane, N. D. (2021, April). μ nas: Constrained neural architecture search for microcontrollers. In *Proceedings of the 1st Workshop on Machine Learning and Systems* (pp. 70-79).
- Fedorov, I., Adams, R. P., Mattina, M., & Whatmough, P. (2019). Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers. *Advances in Neural Information Processing Systems*, 32.
- Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., ... & Yih, W. T. (2020, November). Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)* (pp. 6769-6781).
- Qiu, L. (2025). Reinforcement Learning Approaches for Intelligent Control of Smart Building Energy Systems with Real-Time Adaptation to Occupant Behavior and Weather Conditions. *Journal of Computing and Electronic Information Management*, 18(2), 32-37.
- Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019, June). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)* (pp. 4171-4186).
- Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., & Smith, V. (2020). Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, 2, 429-450.
- Kairouz, P., & McMahan, H. B. (2021). Advances and open problems in federated learning. *Foundations and trends in machine learning*, 14(1-2), 1-210.
- Chen, J., Liu, J., Liang, Y., & Zhou, M. (2026). KE-MLLM: A Knowledge-Enhanced Multi-Sensor Learning Framework for Explainable Fake Review Detection. *Applied Sciences*, 16(6), 2909.