International Journal of Artificial Intelligence, Data Science, and Machine Learning
Grace Horizon Publication | Volume 7, Issue 2, 106-111, 2026
ISSN: 3050-9262 | https://doi.org/10.63282/3050-9262.IJAIDSML-V7I2P116

Original Article

Architectures for Low-Latency AI Inference on Streaming Enterprise Data

Stewyn Chaudhary
Independent Researcher, USA.

Received On: 08/03/2026 ***Revised On: 02/04/2026*** ***Accepted On: 09/04/2026*** ***Published On: 20/04/2026***

Abstract *- Enterprise data streams now demand AI inference systems that deliver low-latency predictions without sacrificing model accuracy. This paper presents a literature-based comparative study of four architectural patterns for machine-learning inference on high-throughput enterprise streams: in-stream inference (ISI), microservice-based pipelines (MIP), co-located model serving (CMS), and edge-offloaded inference (EOI). Each pattern is evaluated on end-to-end latency, throughput, resource utilization, and operational complexity across three representative workloads: financial fraud detection, IoT telemetry anomaly detection, and real-time recommendation. Particular attention is given to three model-optimization techniques post-training quantization, structured pruning, and knowledge distillation as the primary instruments for latency reduction. Synthesizing published benchmarks, the analysis shows that in-stream inference combined with distilled, INT8-quantized models can achieve materially lower end-to-end latency than REST-based microservice baselines, with accuracy losses typically of 1–2 percentage points. A comparative analysis of failure modes across patterns is presented, together with a pattern-selection decision matrix and design guidelines for practitioners building production AI inference systems on Apache Kafka, Apache Flink, and equivalent cloud-native streaming platforms.*

Keywords *- Low-Latency Inference, Streaming Data, Model Optimization, Post-Training Quantization, Knowledge Distillation, Structured Pruning, Enterprise AI, Apache Flink, Apache Kafka, Real-Time Machine Learning, Edge Inference, Failure Modes, Operational Risk.*

1. Introduction

Modern enterprise systems generate continuous, high-volume data streams across virtually every sector. Financial institutions process millions of payment events per second; industrial IoT deployments emit sensor readings at kilohertz frequencies; and e-commerce platforms capture clickstream events in real time. Extracting actionable intelligence from these streams within tight latency budgets is no longer an academic exercise it is a competitive and, in regulated industries, a legal imperative [1].

Traditional batch-oriented ML workflows in which models are trained offline and predictions are served on pre-aggregated feature sets—cannot meet the sub-100 ms service-level objectives (SLOs) imposed by fraud detection, predictive maintenance, and personalization workloads. Streaming architectures push computation closer to the data source, fundamentally reshaping the design space for AI inference [2]. Reducing inference latency, however, is not simply a matter of faster hardware: model complexity, inter-operator scheduling, serialization costs, and network round-trip times each contribute measurably to the end-to-end budget. Model-optimization techniques quantization, pruning, and knowledge distillation have therefore emerged as first-class instruments for latency reduction that any architectural study must address [3].

This paper makes five contributions: (1) a principled classification of four dominant streaming-inference patterns with an explicit latency throughput trade-off analysis; (2) a structured comparison grounded in published benchmarks; (3) a synthesis of published results on quantization, pruning, and distillation within each pattern; (4) a comparative analysis of production failure modes and operational risks; and (5) a pattern-selection decision matrix and design guidelines for practitioners.

2. Related Work

2.1. Stream Processing and Real-Time Analytics

Stream-processing systems such as Apache Kafka Streams [4], Apache Flink [5], and Google Dataflow [6] have matured into production-grade platforms capable of processing tens of millions of events per second with millisecond watermark guarantees. Early ML integrations were ad hoc: models were serialized as user-defined functions (UDFs) and invoked per-event, ignoring the batching opportunities that could amortize GPU kernel-launch overhead. Carbone et al. [5] demonstrated that Flink's dataflow model could be extended with stateful ML operators. Systems such as Clipper [7] and Triton Inference Server [8] subsequently introduced dynamic batching and model ensembles as first-class serving primitives, while TFX [9] addressed model versioning in long-running pipelines.

2.2. Model Optimization for Inference

Post-training quantization (PTQ) converts floating-point weights and activations to lower-bit integer representations

without retraining, achieving 2–4× speedups on commodity hardware with under 1% accuracy loss on standard benchmarks [10]. Quantization-aware training (QAT) recovers additional accuracy at higher training cost [11]. Structured pruning removes entire channels or attention heads, yielding architectures amenable to standard dense matrix operations without sparse-library support [12]. Knowledge distillation [13] trains a compact student model to mimic the soft output distribution of a larger teacher. Task-specific distillation variants have demonstrated further gains in structured prediction tasks [14], [15].

2.3. Low-Latency Serving Architectures

Crankshaw et al. [7] identified model-selection policy and adaptive batching as dominant contributors to tail latency in online serving. Olston et al. [16] documented the operational challenges of serving ML models at scale with TensorFlow Serving, emphasizing the importance of versioning and rollback under continuous deployment. Serverless compute platforms [17] and lightweight virtualization approaches such as Firecracker [18] have been proposed to reduce the cold-start penalties inherent to container-based deployments. None of these works, however, systematically compare architectural patterns in a streaming enterprise context under realistic workload distributions—a gap this paper addresses.

3. Architectural Taxonomy

Four principal deployment patterns exist for AI inference over enterprise data streams, each making distinct trade-offs in latency, throughput, operational complexity, and model-versioning agility. The patterns are not mutually exclusive: production systems increasingly combine two or more within a single pipeline, a topic revisited in Section VII.

3.1. In-Stream Inference (ISI)

In ISI architectures, the inference computation is embedded directly as a stateful operator within the stream-processing directed acyclic graph (DAG). Models are loaded in-process at startup, and events are featurized and scored within the same operator thread, eliminating serialization and inter-process communication overhead. Apache Flink's operator chaining and Kafka Streams' interactive queries are canonical ISI implementations [5].

ISI delivers the lowest end-to-end latency typically in the single-digit millisecond range because no network hop or serialization boundary separates featurization from inference [7]. However, it constrains model complexity to what can execute within operator real-time budgets, making it unsuitable for large transformer or deep CNN models without aggressive compression. Because the operator process is shared with the stream-processing runtime, a model with a memory leak can destabilize an entire streaming topology.

3.2. Microservice-Based Inference Pipeline (MIP)

MIP architectures decouple inference from stream processing via synchronous or asynchronous remote procedure calls (RPCs). The stream processor extracts a feature vector and submits it to a dedicated model-serving microservice such as TensorFlow Serving [16], NVIDIA Triton [8], or Seldon Core; the prediction is then joined back into the stream. MIP provides strong operational isolation: models can be redeployed, A/B tested, and horizontally scaled independently of the stream topology.

The principal cost is the network round trip, which adds tens of milliseconds depending on cluster topology and serialization format. gRPC with Protocol Buffers reduces serialization overhead substantially relative to REST/JSON [19]. Dynamic batching within the serving layer amortizes GPU kernel-launch costs, trading marginal added latency for significantly higher throughput.

3.3. Co-Located Model Serving (CMS)

CMS architectures deploy the inference engine as a sidecar process on the same node (or NUMA domain) as the stream operator. Inter-process communication (IPC) uses shared memory or Unix domain sockets, cutting serialization overhead while preserving process isolation. NVIDIA Triton with shared-memory transport [8] and TorchServe with local IPC are representative CMS implementations.

CMS achieves latencies between those of ISI and MIP while retaining operational independence. The primary challenge is resource contention: the stream operator and inference engine compete for CPU cache, memory bandwidth, and PCIe bandwidth to the GPU. NUMA-aware CPU pinning and memory-bandwidth partitioning are required to avoid throughput degradation under high load [20].

3.4. Edge-Offloaded Inference (EOI)

EOI executes inference on edge devices—industrial PCs, FPGA accelerators, or NPU-equipped gateways—co-located with the data source, forwarding only prediction results to the cloud stream processor. This pattern suits scenarios where raw data volume makes cloud ingestion cost-prohibitive, or where data-sovereignty regulations prohibit raw-data transmission.

EOI offers the lowest network-dependent latency for source-proximate decisions but introduces model heterogeneity: TFLite, ONNX Runtime, and OpenVINO have divergent operator support, complicating model governance across a distributed device fleet. INT8 and binary-neural-network variants enable deployment on memory-constrained edge processors; published benchmarks for MobileNetV2 [21] report sub-10 ms inference times on representative edge hardware.

4. Model Optimization Techniques

This section surveys the three optimization techniques most relevant to streaming inference, drawing on published results to characterize the latency–accuracy trade-offs attainable within each architectural pattern.

4.1. Post-Training Quantization (PTQ)

PTQ converts a trained floating-point model to a reduced-precision representation—typically INT8 or INT4—by calibrating scale factors on a small representative dataset, without any retraining. ONNX Runtime, TensorRT, and OpenVINO implement PTQ with minimal API changes. For transformer-based models, per-channel quantization of weight tensors combined with dynamic quantization of activations preserves accuracy more effectively than symmetric per-tensor schemes [22].

Published evaluations report that INT8 PTQ typically reduces inference latency by 1.5–3× on CPU and 1.3–2.5× on GPU relative to FP32 baselines, with accuracy degradation generally below 1% on classification tasks [10]. PTQ is particularly well-suited to ISI and CMS, where a compact model footprint directly enables in-process or sidecar deployment that would otherwise exceed available memory budgets.

4.2. Structured Pruning

Structured pruning removes contiguous groups of parameters convolutional filters, attention heads, or entire transformer layers whose removal maps cleanly to standard dense matrix operations, so the compressed model can execute without specialized sparse-matrix library support. Established approaches include L1-magnitude filter pruning [12] and attention-head importance scoring [23].

Published results indicate that 50% structured pruning typically yields 1.5–2× throughput improvements at a cost of 1–2 percentage points of accuracy, depending on task and model architecture [12]. Pruning ratios above 60–70% tend to exceed practical accuracy thresholds for high-stakes tasks such as fraud detection, underscoring the need for task-adaptive ratios. Structured pruning is most beneficial in MIP and CMS deployments, where the model server can be updated independently of the stream topology.

4.3. Knowledge Distillation

Knowledge distillation (KD) trains a compact student model to reproduce the soft output distributions of a larger teacher, transferring both task knowledge and calibrated uncertainty [13]. Widely adopted instances include DistilBERT [24]—a 40% smaller, 60% faster student of BERT-base that retains 97% of benchmark performance and task-specific variants applying temperature-scaled logit matching for structured prediction [14].

Distilled models typically achieve 2–4× lower inference latency than their teachers, with accuracy within 1–2 percentage points on held-out test sets [15], [24]. Distillation can reduce a model to a footprint small enough for ISI operator deployment—a reduction often unachievable through quantization or pruning alone. The combined distillation + PTQ pipeline consistently yields the best latency–accuracy Pareto frontier reported in the literature.

5. Comparative Analysis

This section synthesizes the architectural and optimization analyses into a structured comparison across three representative enterprise workloads: financial fraud detection (FFD, SLO ≤ 50 ms), IoT telemetry anomaly detection (ITAD, SLO ≤ 100 ms), and real-time recommendation (RTR, SLO ≤ 200 ms).

5.1. Latency and Throughput

Table 1 summarizes the qualitative latency and throughput characteristics of each architectural pattern, informed by published benchmarks [5], [7], [8], [20].

Table 1: Qualitative Comparison of Streaming Inference Architectural Patterns

Pattern	p50 Latency	Throughput	Op. Complexity	Primary Constraint
ISI	1–10 ms	High	Low	Model size / memory budget
MIP (REST)	20–80 ms	Medium	Medium	Network round-trip; serialization
MIP (gRPC)	10–50 ms	Med.–High	Medium	Network round-trip
CMS	2–15 ms	High	Medium	NUMA contention; PCIe bandwidth
EOI	1–8 ms	Device-bound	High	Edge runtime heterogeneity

5.2. Impact of Model Optimization

Table 2 summarizes the latency reduction and accuracy trade-offs of each optimization technique as reported in the published literature. Accuracy change is expressed as percentage-point (pp) deviation from the full-precision baseline.

Table 2: Model Optimization: Published Latency and Accuracy Trade-Offs

Technique	Latency Reduction	Accuracy Loss	Best-Fit Pattern
PTQ INT8	1.5–3× CPU; 1.3–2.5× GPU	<1 pp	ISI, CMS
Structured Pruning (50%)	1.3–1.7×	1–2 pp	MIP, CMS
Knowledge Distillation	2–4×	1–2 pp	ISI, EOI
Distillation + PTQ INT8	3–5× (combined)	1–2 pp	ISI

The combined distillation + PTQ pipeline offers the most favorable latency–accuracy trade-off, enabling models previously confined to MIP deployments to be embedded directly as ISI operators. For tasks where even 1–2 pp of accuracy loss is unacceptable, MIP with gRPC and dynamic batching remains the preferred pattern, as it accommodates full-precision models without the memory constraints of in-process deployment.

5.3. Failure Modes and Operational Risks

Architectural choice shapes not only steady-state performance but also the failure modes a system must tolerate. Table 3 summarizes the principal operational risks associated with each pattern, drawn from production incidents documented in the literature [7], [9], [16], [18], [26].

Table 3: Failure Modes and Operational Risks by Architectural Pattern

Pattern	Primary Failure Mode	Blast Radius	Mitigation
ISI	Memory leak in model operator	Entire streaming topology crashes	Memory-bounded loading; restart policies
MIP	Serving overload or cold-start spike	Increased tail latency; SLO breach	Autoscaling with pre-warmed replicas
CMS	Resource contention with sidecar	Throughput degradation on co-located workloads	NUMA pinning; cgroup isolation
EOI	Stale model on disconnected device	Silent accuracy degradation in production	Federated versioning; drift monitoring

Three observations stand out. First, ISI carries the highest blast radius per failure: because the model runs inside the stream-processing runtime, a model defect can bring down an entire pipeline topology rather than an isolated service. Thorough offline memory profiling and robust operator restart policies are therefore essential prerequisites for ISI deployment. Second, MIP's principal risk is tail-latency amplification under load spikes. Because synchronous RPC calls block the stream operator thread, a slow serving replica creates backpressure that propagates upstream; circuit breakers and pre-warmed replica pools [7] are the standard mitigations. Third, EOI introduces a uniquely insidious failure mode—silent accuracy drift—because a device that loses connectivity may continue serving predictions from a stale model without any observable error signal at the cloud level. Federated model-versioning protocols and continuous distribution-shift monitoring at the edge are therefore critical for EOI deployments at scale [26].

6. Design Guidelines

The following guidelines synthesize the comparative analysis—including the failure-mode analysis of Section V-C—into actionable recommendations. Practitioners are encouraged to use Table IV as a first-pass pattern-selection tool, then consult the SLO-tiered guidance below to refine their choice.

Table 4: Pattern-Selection Decision Matrix

Criterion	ISI	MIP	CMS	EOI	Notes
SLO < 20 ms	✓✓	✗	✓	✓✓	ISI/EOI viable
SLO 20–100 ms	✓	✓✓	✓✓	—	CMS preferred w/ GPU
Model size > 500 MB	✗	✓✓	✓	✗	Large models need MIP/CMS
Data sovereignty	✓	✓	✓	✓✓	EOI keeps data on-prem
Frequent model updates	✗	✓✓	✓✓	✗	ISI needs operator restart
High fault tolerance	✗	✓✓	✓	✓	MIP isolates failures

- SLO ≤ 20 ms (e.g., fraud detection): Prefer ISI with distilled, PTQ-optimized models. Profile model memory footprint before embedding; a student model that fits within L3 cache eliminates cold-start latency spikes at operator restart. Enforce operator-level memory limits and automatic restart policies to contain the blast radius of model failures.
- SLO 20–100 ms (e.g., IoT alerting, personalization): Prefer MIP with gRPC, dynamic batching, and pre-warmed replicas. Apply structured pruning to reduce serving-instance count and infrastructure cost. Use Protocol Buffers instead of JSON to cut serialization overhead [19]. Deploy circuit breakers between the stream operator and the serving microservice to contain backpressure under load spikes.
- GPU large-model serving (SLO ≤ 100 ms): CMS with shared-memory IPC and INT8 quantization offers the best throughput-to-cost ratio. Apply NUMA-aware CPU pinning and cgroup-based memory isolation to prevent resource contention between the stream operator and the inference sidecar [20].
- Data-sovereignty or bandwidth-constrained deployments: EOI with INT4-quantized models on NPU-equipped gateways. Design a federated model-update and versioning protocol from the

outset; ad hoc fleet governance does not scale beyond a few hundred devices. Deploy continuous distribution-shift monitoring at the edge to detect silent model staleness [26].

Across all patterns, model optimization should be treated as a first-order architectural concern rather than a post-hoc refinement. The recommended workflow is: (1) establish the SLO and workload profile; (2) consult Table IV for a candidate pattern; (3) apply distillation to meet the model-size budget; (4) apply PTQ to reach the target latency; (5) validate accuracy on production-representative held-out data; and (6) define failure-mode mitigations before going live.

7. Discussion

7.1. Scope and Limitations

This paper is a literature-based comparative analysis rather than an original empirical study. Quantitative claims are drawn from published benchmarks and should be read as indicative ranges, not guaranteed outcomes for any specific deployment. Real-world latency and throughput vary with hardware generation, network topology, workload distribution, and system configuration; practitioners should validate the latency characteristics of a chosen pattern against representative production traffic before committing to an architecture.

The workloads considered fraud detection, IoT anomaly detection, and recommendation are structured tabular and sequence-classification tasks. Computer-vision and generative inference workloads may exhibit substantially different latency–throughput profiles. Extending this framework to large-language-model (LLM) inference over live document streams such as retrieval-augmented generation (RAG) is a natural direction for future work, since the memory and compute demands of autoregressive generation introduce fundamentally different latency bottlenecks.

7.2. Hybrid Patterns and Migration Paths

The four patterns are not mutually exclusive; production deployments frequently combine them at different stages of the same pipeline. A common hybrid pairs ISI for lightweight feature-based pre-filters (e.g., rule-based anomaly scoring) with MIP for heavier deep-learning models invoked only on candidate events flagged by the pre-filter. This two-stage design can reduce the number of deep-model inference calls by one to two orders of magnitude, substantially improving overall throughput and cost without compromising detection quality.

Migration between patterns is also a practical concern. Teams often begin with MIP for its operational simplicity and migrate toward ISI or CMS as SLOs tighten and models are compressed through distillation. The reverse migration from ISI to MIP becomes necessary when model complexity outgrows in-process memory budgets. In either direction, the critical dependency is model compatibility: the serving runtime on the target pattern must support the model's operator set and data types, a constraint that makes framework-agnostic formats such as ONNX valuable portability intermediates.

7.3. Sustainability

Model optimization carries a meaningful sustainability co-benefit. Quantized and distilled models perform fewer floating-point operations per inference, reducing energy consumption proportionally. Energy-profiling studies indicate that INT8-quantized models can achieve roughly 2–4× lower energy use per inference than their FP32 counterparts [25]. This is increasingly relevant to enterprise sustainability reporting under emerging disclosure frameworks and should be weighed alongside latency and cost in architecture selection.

8. Conclusion

This paper offered a literature-based comparative study of four architectural patterns for low-latency AI inference on streaming enterprise data, with emphasis on model optimization as the primary latency-reduction instrument and production failure modes as a practical constraint on pattern selection. The analysis indicates that in-stream inference combined with knowledge distillation and INT8 post-training quantization provides the most favorable latency–accuracy trade-off for latency-critical workloads, whereas microservice-based architectures remain preferable when operational flexibility, fault isolation, and model accuracy are the dominant concerns.

The central insight is that architecture selection and model optimization are not independent decisions: the chosen architecture determines the memory and latency budget within which optimization must operate, and the achievable compression ratio determines which architectures are feasible for a given SLO. The pattern-selection matrix and design guidelines derived from this analysis provide practitioners with a systematic framework for navigating this joint design space, including the failure-mode mitigations that are often overlooked at the architecture-selection stage.

Future work should prioritize empirical validation of these comparative findings across diverse hardware configurations, adaptive batching policies that respond dynamically to real-time latency telemetry, federated model-update protocols for EOI device fleets, and extension of the framework to generative inference workloads such as retrieval-augmented generation over live document streams.

Acknowledgment

The author thanks colleagues and advisors who provided feedback on earlier drafts of this work. No external funding was received for this research.

References

[1] M. Armbrust *et al.*, "Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, Houston, TX, USA, 2018, pp. 601–613.

[2] T. Akidau *et al.*, "The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing," *Proc. VLDB Endow.*, vol. 8, no. 12, pp. 1792–1803, Aug. 2015.

[3] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A Survey of FPGA-Based Neural Network Inference Accelerators," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 12, no. 1, pp. 1–26, Mar. 2019.

[4] N. Narkhede, G. Shapira, and T. Palino, *Kafka: The Definitive Guide*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2021.

[5] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache Flink: Stream and Batch Processing in a Single Engine," *Bull. IEEE Comput. Soc. Tech. Comm. Data Eng.*, vol. 36, no. 4, pp. 28–38, Dec. 2015.

[6] T. Akidau *et al.*, "MillWheel: Fault-Tolerant Stream Processing at Internet Scale," *Proc. VLDB Endow.*, vol. 6, no. 11, pp. 1033–1044, Aug. 2013.

[7] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A Low-Latency Online Prediction Serving System," in *Proc. 14th USENIX Symp. Networked Syst. Des. Implement. (NSDI)*, Boston, MA, USA, 2017, pp. 613–627.

[8] NVIDIA Corporation, "Triton Inference Server," NVIDIA Developer Documentation. [Online]. Available: https://developer.nvidia.com/triton-inference-server

[9] D. Baylor *et al.*, "TFX: A TensorFlow-Based Production-Scale Machine Learning Platform," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining (KDD)*, Halifax, NS, Canada, 2017, pp. 1387–1395.

[10] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A Survey of Quantization Methods for Efficient Neural Network Inference," in *Low-Power Computer Vision*, Boca Raton, FL, USA: Chapman & Hall/CRC, 2022, pp. 291–326.

[11] B. Jacob *et al.*, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, USA, 2018, pp. 2704–2713.

[12] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning Filters for Efficient ConvNets," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Toulon, France, Apr. 2017.

[13] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," arXiv:1503.02531, Mar. 2015.

[14] W. Park, D. Kim, Y. Lu, and M. Cho, "Relational Knowledge Distillation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Long Beach, CA, USA, 2019, pp. 3967–3976.

[15] L. Hou, Z. Huang, L. Shang, X. Jiang, X. Chen, and Q. Liu, "DynaBERT: Dynamic BERT with Adaptive Width and Depth," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 33, 2020, pp. 9782–9793.

[16] C. Olston *et al.*, "TensorFlow-Serving: Flexible, High-Performance ML Serving," arXiv:1712.06139, Dec. 2017.

[17] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the Cloud: Distributed Computing for the 99%," in *Proc. ACM Symp. Cloud Comput. (SoCC)*, Santa Clara, CA, USA, 2017, pp. 445–451.

[18] A. Agache *et al.*, "Firecracker: Lightweight Virtualization for Serverless Applications," in *Proc. 17th USENIX Symp. Networked Syst. Des. Implement. (NSDI)*, Santa Clara, CA, USA, 2020, pp. 419–434.

[19] Google LLC, *Protocol Buffers Developer Guide*. [Online]. Available: https://protobuf.dev

[20] Z. Bai, Z. Zhang, Y. Zhu, and X. Jin, "PipeSwitch: Fast Pipelined Context Switching for Deep Learning Applications," in *Proc. 14th USENIX Symp. Oper. Syst. Des. Implement. (OSDI)*, Nov. 2020, pp. 499–514.

[21] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, USA, 2018, pp. 4510–4520.

[22] X. Wei, R. Gong, Y. Li, X. Liu, and F. Yu, "QDrop: Randomly Dropping Quantization for Extremely Low-Bit Post-Training Quantization," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2022.

[23] E. Voita, D. Talbot, F. Moiseev, R. Sennrich, and I. Titov, "Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned," in *Proc. 57th Annu. Meet. Assoc. Comput. Linguist. (ACL)*, Florence, Italy, 2019, pp. 5797–5808.

[24] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter," arXiv:1910.01108, Oct. 2019.

[25] K. Lottick, S. Susai, S. A. Friedler, and J. P. Wilson, "Energy Usage Reports: Environmental Awareness as Part of Algorithmic Accountability," in *NeurIPS Workshop on Tackling Climate Change with ML*, Vancouver, BC, Canada, Dec. 2019.

[26] T. Akidau, S. Chernyak, and R. Lax, *Streaming Systems: The What, Where, When, and How of Large-Scale Data Processing*. Sebastopol, CA, USA: O'Reilly Media, 2018.