Navigating the Optimization Gap: Challenges and Improvements in Deep Learning Model Deployment

LOVINA DMELLO*, Nvidia Corporation, USA
BLAISE DMELLO, University of Washington, USA
LINNET TUSCANO, Syracuse University, USA

Background: Deep learning models that excel on offline benchmarks often underdeliver under strict latency, cost, and reliability targets. We call this mismatch the *optimization gap*.
Objectives: Inference already dominates ML infrastructure budgets, yet datacenter GPUs often sit idle between bursts of traffic, so the gap has a direct cost. We aim to unify the fragmented literature on deployment challenges into a framework that practitioners and researchers can act on.
Methods: Following an explicit search protocol, we review 93 papers, most published since 2018, on model conversion, precision tuning, compiler optimization, inference serving, and operational pipelines across GPUs, TPUs, NPUs, FPGAs, CPUs, and custom cloud ASICs. We decompose the optimization gap into five analytically distinct dimensions: performance, efficiency, compatibility, observability, and operational robustness, the last of which includes security, privacy, and compliance. We then map mitigations at the model, compiler, system, and pipeline levels.
Results: Case studies in vision, large language models, edge deployment, and non-Nvidia hardware ecosystems show how workload characteristics reshape which bottleneck dominates and which interventions pay off most.
Conclusions: Five research directions need substantially more attention: automating optimization policies, ensuring cross-hardware portability across heterogeneous accelerators, restoring observability to compiled graphs, building integrated deployment benchmarks, and treating deployment-time security and compliance as first-class concerns.

JAIR Track: Surveys

JAIR Associate Editor: Pablo Samuel Castro

JAIR Reference Format:
Lovina Dmello, Blaise Dmello, and Linnet Tuscano. 2026. Navigating the Optimization Gap: Challenges and Improvements in Deep Learning Model Deployment. *Journal of Artificial Intelligence Research* 87, Article 26 (November 2026), 27 pages. DOI: 10.1613/jair.1.22113

1 Introduction

A model can top every accuracy leaderboard and still fall apart in production. Latency budgets, throughput floors, and cost ceilings, all constraints absent from standard training losses, expose weaknesses that offline evaluations often overlook. We refer to this mismatch as the *optimization gap*. To avoid the ambiguity that a loosely used term invites, we fix a precise definition and hold to it for the remainder of the paper.

*Corresponding Author.

Authors' Contact Information: Lovina Dmello, ORCID: 0009-0002-5603-0594, ldmello@nvidia.com, Nvidia Corporation, Santa Clara, California, USA; Blaise Dmello, ORCID: 0009-0008-0234-4743, blaisedmello2002@gmail.com, University of Washington, Seattle, Washington, USA; Linnet Tuscano, ORCID: 0009-0000-7259-0001, linnettuscano@gmail.com, Syracuse University, Syracuse, New York, USA.



This work is licensed under a Creative Commons Attribution International 4.0 License.

© 2026 Copyright held by the owner/author(s).
DOI: 10.1613/jair.1.22113

> **Definition (Optimization gap).** The *optimization gap* is the measurable degradation a deep learning model incurs between (i) its offline benchmark configuration, in which accuracy is measured on curated data under idealized batching on a reference device, and (ii) its steady-state production deployment under real latency, cost, reliability, and governance constraints. We decompose this degradation into five orthogonal dimensions (*performance*, *efficiency*, *compatibility*, *observability*, and *operational robustness*), each defined by the specific production property it degrades (§4, Table 2).

Throughout, every use of "optimization gap" refers to one or more of these five dimensions, never to a vague notion of underperformance (Definition above). This usage is distinct from the *optimization gap* in neural architecture search, which denotes instability between a shared super-network and its sub-architectures rather than deployment degradation [L. Xie et al. 2022]. When we distinguish a specific facet, for example a throughput drop versus data drift, we name the dimension explicitly so that the reader can trace each claim to a measurable production property.

The financial stakes alone justify attention. Inference accounts for over 90% of ML infrastructure spending on AWS [Romero et al. 2021], and this share is widely expected to grow as AI features approach web-search-scale traffic [Fu et al. 2024]. A large share of that spending is wasted. GPUs powering inference go idle between request bursts [Z. Li et al. 2023; Yu et al. 2022], but consolidating models onto shared hardware to recover utilization introduces a different problem: resource contention that produces unpredictable latency spikes and violates service-level agreements (SLOs) [F. Xu et al. 2023].

Figure 2 shows that the gap is not confined to one layer; it threads through the entire deployment stack. ONNX converters silently introduce numerical defects that yield wrong outputs without raising errors [Jajal et al. 2024]. Quantization gains erode once calibration data drifts from what the model encounters in production [Hubara et al. 2021; Moon et al. 2024]. Serving frameworks must balance accuracy, cost, and speed under volatile demand [Ahmad et al. 2024; Razavi et al. 2024; Salmani et al. 2023]; hardware-specialized serving can boost throughput by an order of magnitude, but only for workloads that fit the optimized path [Wei et al. 2022]. On the operations side, the MLOps discipline needed to govern deployed models has not kept pace. Simply reproducing an inference environment without introducing regressions remains a live problem in both industry and research [Brewer et al. 2021; Nellutla 2026; Olszewski et al. 2023; Y. Wu et al. 2022].

This survey charts these interacting failure points and investigates why the same breakdowns recur at different stack layers. We contribute the following:

(1) A *taxonomy of the optimization gap* that decomposes deployment failures into five analytically distinct but interacting dimensions (performance, efficiency, compatibility, observability, and operational robustness), gives each an explicit discriminating criterion (Table 2), and maps each to concrete intervention layers (§4).
(2) A *cross-layer analysis of mitigation strategies* that synthesizes, rather than merely catalogs, why particular approaches generalize across workloads and hardware while others remain locked to a single target (§5).
(3) A critical assessment of *evaluation and benchmarking pitfalls* that explains why current offline metrics poorly predict production behavior (§6).
(4) *Cross-domain case studies* in vision, LLM, edge, and heterogeneous (non-Nvidia) hardware workloads (§7) that reveal how workload characteristics and hardware backends shift the binding constraint across the taxonomy, together with a dedicated treatment of *security, privacy, robustness, and compliance* in deployment (§8), practitioner best practices, and open research directions (§9).

We organize the discussion around three questions: Why do the same deployment failures keep surfacing at different stack layers? Which optimization strategies transfer across hardware, and which are locked to a single target? How should evaluations be structured so that their results predict production outcomes rather than benchmark rankings alone?

Table 1. Scope comparison with related surveys. ✓ = primary focus; ~ = partial coverage; — = not addressed.

Survey	Model Opt.	Compiler	Serving	MLOps	Benchmarking
[Zhou et al. 2024]	~	~	✓	—	—
[B. Li et al. 2024]	—	—	✓	—	~
[M. Li et al. 2021]	—	✓	—	—	—
[Yu et al. 2022]	—	—	✓	—	—
[H. Xia et al. 2024]	✓	—	~	—	—
[Shi et al. 2024]	✓	—	~	—	—
This survey	✓	✓	✓	✓	✓

A note on evidence versus advice. Survey readers are ill-served when reported findings and the authors' own recommendations blur together. We therefore adopt a strict convention: unlabeled statements summarize results reported in the cited literature, whereas any forward-looking guidance we offer to practitioners is set off explicitly with the marker ***Practitioner takeaway.*** This lets the reader separate what the evidence shows from what we advise engineers to do with it.

2 Scope and Related Work

Our scope spans the entire post-training lifecycle: model export and conversion (e.g., ONNX, TensorRT, TVM, XLA), efficiency techniques (quantization, pruning), scalable inference architectures (dynamic batching), MLOps practices (CI/CD, profiling), and the deployment-time security, privacy, and compliance concerns that increasingly govern production AI. We consider these across a deliberately heterogeneous set of hardware backends (GPUs, Google TPUs, mobile and embedded NPUs, FPGAs, multicore CPUs, and custom cloud ASICs) and their associated compiler ecosystems, rather than a single vendor's stack. Training algorithms and theoretical machine learning fall outside this scope because they do not directly influence the deployment pipeline. Throughout the paper, **latency** means the total request-to-response time, with particular attention to tail latency (p99); **throughput** is the number of requests served per unit time; and **portability** captures how easily a model transfers to a new hardware target without sacrificing accuracy or speed.

Several existing surveys examine individual layers of this stack, but none trace the optimization gap end to end. Zhou et al. [2024] cover foundation-model infrastructure broadly yet pay limited attention to compiler passes and export-time failures. B. Li et al. [2024] focus on LLM serving, including continuous batching and KV-cache management, without connecting those system-level advances to model-level or operational concerns. M. Li et al. [2021] go deep on compiler internals but stop at the compiler boundary, omitting model export upstream and serving and MLOps downstream. Other surveys target narrower sub-problems: multi-tenant GPU sharing [Yu et al. 2022], edge AI [Shankar 2024], speculative decoding [H. Xia et al. 2024], and KV-cache compression [Shi et al. 2024]. We fill this gap with a taxonomy (Figure 3) that spans five dimensions and maps each to concrete intervention layers (Table 1).

3 Methodology

To make the corpus reproducible rather than opportunistic, we followed an explicit search-and-selection protocol adapted from established guidelines for systematic literature studies [Wohlin 2014].

Databases and dates. We searched four databases (Google Scholar, IEEE Xplore, the ACM Digital Library, and arXiv), restricting results to work published or first posted between 1 January 2018 and 28 February 2026. The main search was run in February 2026; two supplementary searches, run in July and October 2026 with the same

databases and criteria, targeted heterogeneous and custom accelerators and deployment-time security. We chose 2018 as the lower bound because it precedes both the transformer-driven surge in serving-system research and the maturation of production ML compilers; a small number of foundational security and hardware references predating 2018 were admitted through snowballing (below) because no later equivalent exists.

Search terms. Searches were keyword-based and targeted at the topics this survey covers: we maintained a search plan of 38 query strings, each tied to a section of the paper (for example, taxonomy, mitigation strategies, evaluation pitfalls, or one of the case studies), and many restricted to top systems and ML venues such as MLSys, OSDI, NSDI, ASPLOS, NeurIPS, ICML, and ICLR, or to IEEE and ACM journals. Each query was the Boolean conjunction of a *task* term and a *deployment* term. Task terms included `"deep learning inference"`, `"model deployment"`, `"LLM serving"`, and `"neural network compiler"`. Deployment terms included `quantization`, `pruning`, `"operator fusion"`, `"dynamic batching"`, `"KV cache"`, `"speculative decoding"`, `"tail latency"`, `"serverless inference"`, `MLOps`, `"model conversion"` / `ONNX`; the supplementary searches added `TPU`, `FPGA`, `NPU`, `"ROCm"` / `AMD`, `"membership inference"`, `"model extraction"`, `"differential privacy"`, and `"inference accelerator"` / `ASIC`.

Inclusion and exclusion criteria. We *included* a paper if it (i) concerned the post-training deployment lifecycle, (ii) reported quantitative evidence on inference performance, efficiency, portability, observability, operations, or security, and (iii) was peer-reviewed or a high-quality preprint with a clear methodology. Among otherwise eligible papers, we gave priority to work in peer-reviewed top-tier venues and journals. We *excluded* work that (a) addressed only training-time algorithms or theory, (b) reported no measurements on real or simulated hardware, (c) was superseded by a later, more complete version by the same authors, or (d) duplicated an already-included result without added deployment insight.

Snowballing and result. We supplemented database search with backward and forward snowballing from key benchmarking and serving surveys [B. Li et al. 2024; Reddi, Cheng, et al. 2020; Zhou et al. 2024], iterating until no new in-scope work surfaced. In the main pass we read 88 candidate papers in full; applying the criteria above to these candidates and to snowballed work produced an initial corpus of 80 papers. The supplementary searches added 11 papers, chiefly on non-Nvidia accelerators and their compiler ecosystems [Chen et al. 2018; Jouppi, Kurian, et al. 2023; Jouppi, Young, et al. 2017; Lattner et al. 2021; Umuroglu et al. 2017] and on deployment-time security, privacy, and robustness [Abadi et al. 2016; Goodfellow et al. 2015; Shokri et al. 2017; Tramèr and Boneh 2019; Tramèr, F. Zhang, et al. 2016], and then 2 papers on custom and dataflow accelerators [Emani et al. 2022; Firoozshahian et al. 2023]. The final corpus contains 93 papers (Figure 1, Appendix A); each included work is cited at least once in the body, and the appendix lists the complete set for reproducibility. We did not log the number of records returned by each query or screened by title and abstract, so we cannot report a full PRISMA-style flow; the counts above are those we can reconstruct from our search plan and full-text collection.

4 A Taxonomy of the Optimization Gap

Analyzing the 93 papers in our corpus, we distilled recurring failure patterns into five dimensions: performance, efficiency, compatibility, observability, and operational robustness. The dimensions are not a loose grouping of symptoms; each is pinned to a single *discriminating question* and a distinct family of production metrics, so that any given failure can be assigned to exactly one dimension *by its root cause*, even when its symptoms spill across several (Table 2). This root-cause rule is what keeps the boundaries analytically sharp. A compatibility bug that first appears as a latency regression, for instance, still belongs to the compatibility dimension because its defining question, “does the model still compute the correct function on this target?”, is the one that fails; the latency symptom is downstream. We use this rule consistently: performance concerns *speed as experienced per request*; efficiency concerns *resource cost per unit of work*; compatibility concerns *numerical correctness after*

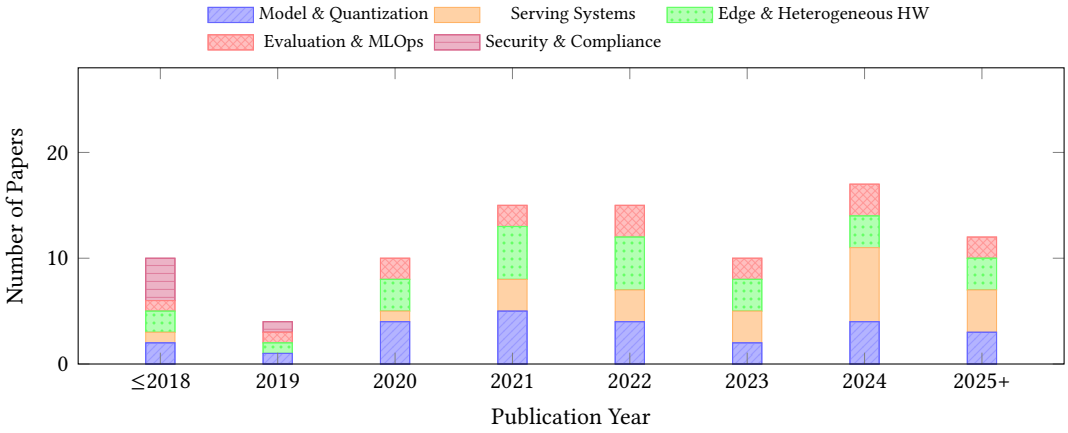

Fig. 1. Distribution of the 93 surveyed papers by publication year and topic category; each paper is counted once, under its primary topic, so category totals are indicative rather than a multi-label classification. Serving-system and LLM-related work grew markedly from 2022 onward, while model-level optimization and edge/heterogeneous-hardware research remained steady. Foundational security and privacy references predate the main window and are grouped in the ≤2018 bucket.

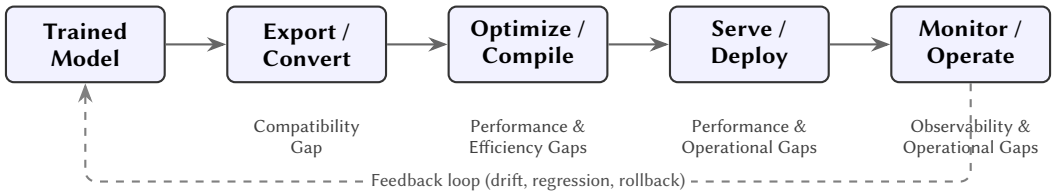

Fig. 2. End-to-end deployment pipeline with optimization gap dimensions annotated at each stage. The dashed feedback loop represents operational responses, such as rollback, retraining, or recalibration, triggered by monitoring.

a target change; observability concerns *our ability to attribute a symptom to a cause*; and operational robustness concerns *the release-and-runtime process surviving change, drift, and adversarial conditions.*

As Figure 3 illustrates, the dimensions remain coupled in *manifestation* even though they are orthogonal in *definition*: poor observability can mask efficiency drains for months, and a single root cause frequently propagates across layers. The value of the taxonomy is precisely that it lets a practitioner trace a tangled symptom back to the one dimension whose defining property has broken, and then choose an intervention that targets that property rather than its shadow. The subsections below define each dimension, describe its typical symptoms, and pinpoint the stack layers where the problems originate; deployment-time security, privacy, and compliance are treated as facets of operational robustness and analyzed in depth in §8.

4.1 Performance Gap

Of all deployment failures, performance problems are the loudest: requests time out, dashboards turn red, and users notice immediately. Offline evaluations report averages, such as mean latency and peak throughput, but production SLOs depend on the *worst case*, especially the 99th-percentile tail. Three recurring patterns explain most of the gap between those perspectives.

Table 2. Analytical boundaries of the five optimization gap dimensions. Each dimension is defined by a distinct discriminating question and metric family; failures are assigned by root cause, not by symptom. The right column names the most common dimension a failure is *mistaken* for, clarifying where the boundaries lie.

Dimension	Discriminating question	Primary metrics	Often confused with
Performance	Is each request fast enough, especially at the tail, under realistic traffic?	p95/p99 latency; throughput under bursts	Efficiency (a fast system can still be wasteful)
Efficiency	Is the workload sustainable in resource terms?	$/inference; energy/request; utilization	Performance (low cost can coexist with high latency)
Compatibility	Does the model compute the correct function after export/compilation on the target?	numerical fidelity vs. reference; export success rate	Observability (silent, so mistaken for "no problem")
Observability	Can a symptom be attributed to a cause?	attribution latency; metric-to-model coverage	Performance or compatibility (masks their root causes)
Operational robustness	Does the release-and-runtime process survive change, drift, and attack?	rollback time; gate coverage; drift/attack resilience	Compatibility (process failure vs. numerical failure)

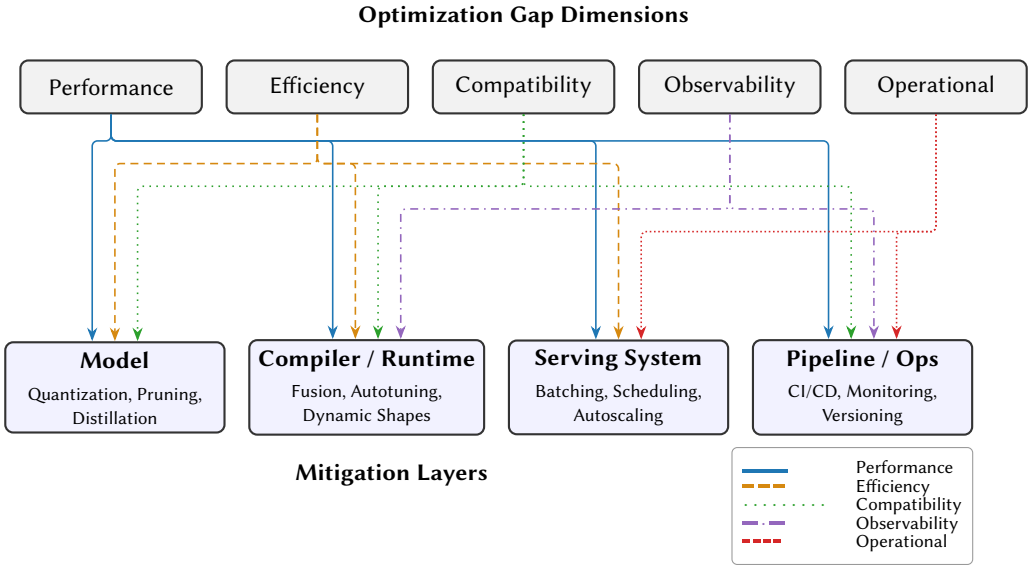

Fig. 3. Overview of the optimization gap taxonomy. The five gap dimensions (top) capture distinct classes of deployment mismatch. Arrows indicate which mitigation layers (bottom) primarily address each dimension. In practice, the dimensions interact: for example, a compatibility issue may manifest as a performance regression, and poor observability can mask efficiency problems.

Latency spikes. Low median latency can coexist with 99th-percentile spikes an order of magnitude worse [Berger et al. 2018; Kasture and Sanchez 2016]. Co-locating models on shared hardware widens the

spread further, injecting interference that single-model profiling never reveals [F. Xu et al. 2023]. Paradoxically, the same optimizations that improve average-case performance, such as larger batches and higher utilization, can *worsen* tail latency [Z. Li et al. 2023], locking efficiency and reliability into a trade-off that parameter tuning alone cannot resolve.

Throughput under realistic traffic. Peak throughput, measured by saturating a GPU with uniform requests, rarely predicts what happens in production, where requests arrive in bursts and vary widely in length. Y. Yang et al. [2022] found that generic serverless architectures could not sustain both low latency and high throughput, and therefore had to be replaced with purpose-built systems. Unless the benchmark reproduces realistic traffic patterns, its throughput numbers are essentially decorative.

Accuracy-latency trade-offs. When demand spikes beyond capacity, serving systems face an uncomfortable choice: breach latency SLOs or quietly sacrifice accuracy by routing requests to a cheaper model variant. Proteus [Ahmad et al. 2024] and InfAdapter [Salmani et al. 2023] make this trade-off explicit through dynamic accuracy scaling, and both report fewer SLO violations than static autoscalers. The implication is clear: accuracy is no longer fixed at deployment time and instead becomes a runtime control that the serving layer adjusts in response to load.

A common thread runs through all three patterns. Average-case optimization is a mature engineering discipline; controlling worst-case behavior under unpredictable multi-tenant load is not, and no general solution has emerged.

4.2 Efficiency Gap

Meeting every latency target is not the same as running a sustainable system. A model can satisfy every SLO and still burn through hardware budgets, drain batteries, or waste GPU-hours at a rate no organization can absorb indefinitely. Where performance failures are immediate (a request times out and a user retries), efficiency failures are slower to surface: costs creep up over days, energy bills rise over weeks, and idle GPUs can go unnoticed for months.

Memory pressure. GPU memory sets a hard ceiling on deployment scale, and LLMs press against it from two directions. Weights alone occupy tens of gigabytes [Yao et al. 2022], but the KV cache, which stores past-token context, often dominates memory use, growing linearly with sequence length until it eclipses the model itself in long conversations [Shi et al. 2024]. Quantization helps reduce footprints [H. Wu et al. 2020], yet solving the capacity problem only exposes the next one: memory *bandwidth* becomes the bottleneck during the decode phase of autoregressive generation [Sanovar et al. 2025].

Energy and cost escalation. Energy consumption is shaped by the interplay of batch size, hardware generation, and clock policy [Jahanshahi et al. 2020]. Compiler-optimized pipelines on edge accelerators can slash energy use [Jeong et al. 2022], but pushing hardware to its limit hits diminishing returns quickly. On mobile processors, for instance, maximizing clock speed burns through battery far faster than it reduces latency [Wang et al. 2021]. The resulting energy-latency curve is nonlinear and hardware-specific, defying one-size-fits-all optimization [Jahanshahi et al. 2020; Wang et al. 2021].

Underutilization. Bursty traffic leaves expensive GPUs idle between request waves, yet sharing those GPUs across models invites the contention and tail-latency degradation described in §4.1 [F. Xu et al. 2023; Yu et al. 2022]. Live request migration [Sun et al. 2024] eases the conflict but does not resolve it: utilization and isolation are structurally opposed, and no amount of scheduling ingenuity can make the trade-off vanish.

Worse, these pressures feed on each other. Cheaper per-request costs invite longer contexts and higher concurrency, which recreate the very memory and contention problems the original optimization was designed to solve [Tornede et al. 2023].

4.3 Compatibility and Portability Gap

Porting a model from its training framework to a new hardware target is rarely as routine as advertised. What makes compatibility failures especially dangerous is that they affect *correctness*, not just speed: a model can produce silently wrong answers on new hardware without any error or warning.

Model conversion failures. Exporting a model to an intermediate format like ONNX is supposed to be a one-line operation, but Jajal et al. [2024] document a steady stream of defects during node conversion. The worst failures are invisible: the exported model runs, returns outputs, and raises no exception, yet those outputs are numerically wrong. Because the resulting accuracy shifts are subtle, they slip past standard validation and surface only after the model is already in production.

Runtime and hardware mismatch. Compilation gains seldom survive a hardware change. TensorRT may deliver impressive speedups on one GPU generation, only for those gains to evaporate on the next [Lindkvist 2024; Shafi et al. 2021]. Broader compiler surveys echo this finding: operator-level mismatches remain a stubborn portability barrier, especially across vendor ecosystems [M. Li et al. 2021]. For most teams, compiled inference artifacts are effectively disposable; each hardware migration requires fresh compilation, profiling, and validation.

Compiler bugs. Metamorphic testing [D. Xiao et al. 2022] has uncovered flaws in standard optimization passes that silently regress accuracy on new hardware. Detecting these bugs is hard precisely because the compiled model runs at full speed and throws no exception; only head-to-head comparison with the reference model exposes the discrepancy. Taken together, hidden conversion errors, hardware-locked compilation, and compiler bugs are documented across independent studies, which indicates that they are routine deployment risks rather than corner cases. ***Practitioner takeaway.*** Any pipeline targeting multiple platforms must budget for cross-platform validation as a first-class engineering task.

4.4 Observability and Debuggability Gap

Compiling and fusing a model's graph for speed comes at a price: the internal structure that engineers rely on for debugging is stripped away. Once that structure is gone, a performance spike (§4.1) and an unnoticed compatibility regression (§4.3) can look identical from the outside: degraded output with no obvious cause, even though they demand entirely different fixes.

Post-conversion regression detection. Quantized or compiled models can react to shifted input distributions in ways nobody anticipated. Shafi et al. [2021] found that certain TensorRT optimizations reduced top-1 error on corrupted-image benchmarks, which the authors attribute to reduced overfitting rather than to formal adversarial robustness; the effect is a beneficial side effect but entirely uncontrolled. If compilation can accidentally change model behavior for the better, it can just as easily change it for the worse. Without systematic regression tests that compare compiled and reference outputs under distribution shift, practitioners often discover these changes only through production incidents.

Profiling limitations. Profilers faithfully report hardware-level kernel metrics, including clock cycles, occupancy, and memory throughput, but after graph fusion, mapping those numbers back to the original model layers is a different matter entirely. Blackthorn [Lechner and Jantsch 2021] offers layer-level attribution for

CNNs on edge devices, yet nothing comparable exists for LLMs, whose prefill and decode phases have fundamentally different compute profiles [Sanovar et al. 2025]. As a result, when latency regresses, practitioners often cannot tell whether the fault lies in the model, the compiler, or the serving stack.

Reproducibility. Reproducing ML results is already hard [Olszewski et al. 2023]; hardware acceleration makes it harder. Compiling the same model on the same GPU with a different driver version can yield a different engine [Shafi et al. 2021]. Pair non-deterministic compilation with sparse logging and the familiar "it worked on my machine" problem extends wholesale from training to inference.

Monitoring gaps. Even well-instrumented serving systems rarely connect infrastructure metrics to model-level health. Frameworks collect copious runtime statistics, yet few wire those signals to real-time accuracy proxies; reference architectures call for unified monitoring with drift and regression gates [Nellutla 2026; Olszewski et al. 2023], but most production pipelines still wall off training monitoring from inference monitoring. This creates a familiar blind spot: teams measure server-side latency to the microsecond yet still lack an automated path from anomaly to model-level cause.

4.5 Operational Gap

A model can be well-designed and cleanly compiled and still fail because the release process around it is broken. Operational failures are organizational, not algorithmic: they stem from gaps in process rather than gaps in technique.

CI/CD fragility. ML pipelines are more fragile than their conventional software counterparts because model quality is statistical. A seemingly benign model update might shave half a percent off accuracy, a regression invisible to unit tests and detectable only through dedicated statistical checks. MLOps blueprints prescribe always-on data-quality gates [Nellutla 2026], but adoption is patchy. Without those gates, a degraded model can slip into production and stay there, undetected, until user complaints finally surface the problem.

Rollback complexity. Rolling back a web service usually means reverting code. Rolling back an ML model requires coordinating model artifacts, feature stores, and configuration state, and getting any one of those wrong can leave the system in an inconsistent state. A large-scale HPC deployment [Brewer et al. 2021] sustained roughly one million inference samples per second across 192 GPUs, yet its rollback process was still manual. In that same system, an FP16 conversion failed silently: the pipeline fell back to FP32 without alerting anyone, illustrating exactly how conversion failures and operational gaps compound one another [Brewer et al. 2021].

Versioning and governance. Serverless architectures [Y. Wu et al. 2022] and automated routing systems [Romero et al. 2021] dynamically swap in the cheapest model variant that meets a latency budget. This convenience creates a governance headache: when the system switches variants transparently, tracing which version produced a given prediction becomes nontrivial. In regulated industries, such as healthcare, finance, and autonomous driving, this lack of prediction lineage is not merely inconvenient; it is a compliance risk.

Security and robustness exposure. A deployed model is also an attack surface, and this too is an operational-robustness concern because it is the surrounding process, not the model's accuracy, that fails. Query interfaces leak: prediction APIs enable model-extraction attacks that reconstruct proprietary weights [Tramèr, F. Zhang, et al. 2016], and confidence outputs enable membership-inference attacks that reveal whether a record was in the training set [Shokri et al. 2017]. Meanwhile, the optimizations that close other gaps can quietly widen this one: quantization and compilation can change a model's decision surface and its behavior on corrupted or adversarial inputs, sometimes for the better but without any guarantee [Goodfellow et al. 2015; Shafi et al. 2021]. Because

Table 3. Summary of mitigation strategies organized by stack layer. Each row lists a representative technique, its primary mechanism, key reported benefit, and main limitation.

Layer	Approach	Representative Work	Key Benefit	Limitation
Model	Post-training quant.	ZeroQuant [Yao et al. 2022]	3.67× speedup (GPT-J 6B)	Accuracy sensitive to calibration
	Quant.-aware training	DEGREE-QUANT [Tailor et al. 2021]	4.7× CPU speedup (INT8)	Requires retraining
	Adaptive pruning	[Shinde 2024]	91.16% acc. at 1.08-bit avg.	Sensitivity analysis scales with depth
	Knowledge distillation	[Haroush et al. 2020]	No calibration data needed	Limited to batch-norm architectures
Compiler / Runtime	Operator fusion	DNNFusion [Niu et al. 2021]	1.4–3.5× over TVM/MNN	Pattern discovery is model-specific
	Kernel autotuning	[C. Xia et al. 2024]	1.1× / 3.1× over TRT/TVM	Requires per-hardware profiling
	Dynamic shapes	TRT opt. profiles [Lindkvist 2024]	Handles variable inputs	Poor profiles negate gains
Serving System	Dynamic batching	TBDB [Gao et al. 2024]	Adapts to request rate	Inflates latency at low traffic
	Statistical multiplexing	AlpaServe [Z. Li et al. 2023]	10× higher request rate	Model-parallel overhead
	Continuous batching	Sarathi-Serve [A. Agrawal et al. 2024]	2.6× capacity (Mistral-7B)	Prefill/decode decoupling complexity
	Serverless	ServerlessLLM [Fu et al. 2024]	10–200× faster cold start	GPU access still limited
Pipeline / Ops	Regression gates	[Nellutla 2026]	Blocks regressed models	Adoption remains uneven
	Version pinning	Codabench [Z. Xu et al. 2022] / DLHub [Chard et al. 2019]	Reproducible artifacts	Manual coordination required
	Deployment benchmarking	[Brewer et al. 2021]	smiBench compares Flask, TF Serving, RedisAI	No accuracy-proxy linkage

these failures are silent and process-dependent rather than accuracy-visible, they belong to the same family as broken rollback and missing gates; we analyze them, together with privacy and compliance mitigations, in §8.

What makes operational failures particularly destructive is their interaction with every other dimension. A missing CI/CD gate lets a compatibility regression reach production; a broken rollback procedure then turns what should be a recoverable incident into an extended outage; and an unmonitored query interface turns a convenience feature into a data-leakage channel.

5 Current Mitigation Strategies

Practitioners and researchers have responded to each dimension of the optimization gap (§4) with mitigations at the model, compiler, serving-system, and pipeline layers. Table 3 catalogs representative approaches alongside their reported benefits and principal limitations. Our focus here is on structural trade-offs and cross-layer interactions; hardware-specific speedup numbers are difficult to compare across studies and are left to the original papers.

5.1 Model-Level Techniques

Model-level techniques change the network itself before it reaches a compiler, trading a controlled loss of accuracy for smaller footprints and cheaper arithmetic.

Quantization. Lower numerical precision means smaller memory footprints and faster inference, but the accuracy cost varies sharply with architecture, calibration quality, and hardware target (Table 4). PTQ with per-channel scaling works well for CNNs [Hubara et al. 2021; H. Wu et al. 2020] and LLMs [Yao et al. 2022]. ViTs, however, resist naive quantization: their softmax and layer-normalization operations amplify rounding errors into structural failures [Liu et al. 2021], and taming those errors demands dynamic, instance-aware calibration [Moon et al. 2024]. Hardware adds another variable: the same INT8 recipe can produce measurably different accuracy impacts across edge platforms [Rachmanto et al. 2024].

When PTQ damages accuracy beyond acceptable limits, quantization-aware training (QAT) offers a more reliable path at the cost of a retraining cycle. QAT is especially useful for architectures with irregular compute graphs, such as GNNs [Tailor et al. 2021]. Recent low-rank weight factorization has made it tractable even for billion-parameter LLMs running on commodity hardware [Bondarenko et al. 2024].

Table 4. Comparison of quantization methods. PTQ = post-training quantization, QAT = quantization-aware training. Speedup and accuracy figures are as reported by each work on the indicated model.

Method	Type	Precision	Target Model	Speedup / Savings	Accuracy Impact
[Hubara et al. 2021]	PTQ	INT8	General CNNs	—	<1% drop (per-channel)
ZeroQuant [Yao et al. 2022]	PTQ	INT8	GPT-J 6B	3.67× over FP16	Negligible
ZeroQuant [Yao et al. 2022]	PTQ	INT8	GPT-NeoX 20B	2× GPU reduction	Negligible
[Liu et al. 2021]	PTQ	INT8	Vision Transformers	—	Significant ViT degradation
[Moon et al. 2024]	PTQ	6/6-bit	ViTs	—	<0.9% drop
[Rachmanto et al. 2024]	PTQ	INT8	CNNs (edge)	55–67% latency reduction	Varies by platform
DEGREE-QUANT [Tailor et al. 2021]	QAT	INT8	GNNs	4.7× on CPU	Accuracy preserved
[Bondarenko et al. 2024]	QAT	Mixed	7B-param LLM	Single-GPU training	Matches full QAT
[Shinde 2024]	Mixed	1.08-bit avg.	VGG19	—	91.16% (≈ FP32 baseline)
NeuronQuant [H. Li et al. 2025]	PTQ	5.2-bit avg.	Spiking NNs	1.51× power reduction	No accuracy loss

Pruning and compression. Pruning promises smaller models, but the gap between theoretical and realized speedup is wide. Unstructured sparsity, which zeroes out individual weights, defeats GPU parallelism; only structured pruning (removing whole channels or layers) translates to wall-clock gains, and it requires per-layer sensitivity analysis to avoid accuracy collapse [Shinde 2024]. Knowledge distillation can claw back the accuracy lost to aggressive pruning [Musa et al. 2025], even when training data is unavailable, by using batch-normalization statistics as a proxy [Haroush et al. 2020]. These techniques extend to ultra-low-power domains as well. Spiking neural networks benefit from neuron-level quantization that leverages their inherently sparse activations [H. Li et al. 2025], and communication-aware model compression further reduces the overhead of distributed edge inference [Bhardwaj et al. 2020].

5.2 Compiler and Runtime Techniques

Compression alone does not guarantee fast execution. Most compressed models still run through default framework paths, such as PyTorch eager mode, that leave much of the hardware's capability unused. Compilers close this gap by lowering high-level operations to hardware-specific code, gaining speed at the expense of portability.

Graph optimization and operator fusion. Merging adjacent operations into a single GPU kernel (operator fusion) eliminates intermediate memory round-trips and is typically the largest source of compiler-driven speedup [M. Li et al. 2021]. Standard compilers rely on hard-coded fusion patterns; algorithmic approaches that classify operator interactions can discover opportunities the patterns miss, matching or exceeding hand-tuned compilers [Boehm et al. 2018; Niu et al. 2021; Zhao et al. 2022]. An important caveat: fusion gains are *model-specific*. Across the studies we surveyed, the same compiler achieves substantial speedups on some architectures and negligible improvement on others.

Kernel selection and autotuning. Empirical profiling, where candidate kernels are benchmarked on the target GPU and the fastest is selected, delivers strong results but binds the model to one architecture [Shafi et al. 2021]. Systematic kernel generation driven by dataflow analysis and structured code-generation strategies is more portable [C. Xia et al. 2024; Zhu et al. 2026], though its coverage of edge architectures is still thin [Ghadani et al. 2020; Verma et al. 2021]. Neither approach eliminates the need for per-hardware validation; compilation remains a recurring cost, not a one-time step.

Dynamic shape and runtime adaptation. Real-world inputs, including variable-length text and irregularly cropped images, clash with compilers built for fixed tensor shapes. TensorRT's "optimization profiles" offer a

workaround, but misconfigured profile boundaries silently push execution onto unoptimized code paths [Lindkvist 2024]. On edge hardware the problem is sharper still: a single static compilation cannot cover the range of input shapes, and runtime frameworks must detect and adapt to shape-induced performance cliffs on the fly [Jeong et al. 2022].

Cross-vendor compiler stacks. Vendor-specific compilers such as TensorRT deliver the highest single-target throughput but are, by construction, the least portable; a body of work instead pursues speed *and* portability through hardware-agnostic intermediate representations. Apache TVM lowers a single high-level graph to CPUs, GPUs, and accelerators via a learned cost model, trading a modest per-target autotuning cost for coverage no single-vendor toolchain offers [Chen et al. 2018]. MLIR generalizes the idea to a reusable, multi-level IR designed to be shared across hardware targets and adopted by production compiler stacks (including components of Google's XLA/OpenXLA ecosystem), letting vendors reuse optimization passes instead of rebuilding them per chip [Lattner et al. 2021]. The synthesis that emerges from comparing these lines of work is that portability is won at the IR boundary, not the kernel boundary: approaches that expose a stable, hardware-neutral representation and defer target-specific choices to a retargetable backend generalize across vendors, whereas approaches that bake in one vendor's kernels or memory model do not, however fast they run on their home target [M. Li et al. 2021; C. Xia et al. 2024].

5.3 Serving-System Techniques

Once a model is compiled and loaded, the serving layer must absorb the full variability of live traffic. GPU utilization and latency SLOs pull in opposite directions, and recent scheduling work tackles this tension at multiple levels (Table 6).

Batching and scheduling. Static batching performs well at steady load and falls apart when traffic fluctuates. Dynamic batching adjusts to request arrival rates [Gao et al. 2024], though during quiet periods the system wastes time waiting for a batch to fill, inflating latency for requests that could have been served immediately [Samarasinghe 2025]. LLMs add a twist: autoregressive generation splits into a compute-heavy prefill phase (encoding the prompt) and a memory-bandwidth-heavy decode phase (generating tokens) [A. Agrawal et al. 2024]. Batching both phases together forces fast decodes to stall behind slow prefills. Phase-decoupled schedulers solve this by routing each phase into an independent stream, unlocking large capacity gains [A. Agrawal et al. 2024]. When even that is insufficient to contain tail latency, live migration of in-flight requests across servers adds another lever [Sun et al. 2024].

Multi-model and multi-tenant serving. Dedicating a GPU to every model is cost-prohibitive, but sharing hardware reintroduces the interference problem. Two design philosophies have emerged. *Multiplexing* exploits traffic diversity: if different models peak at different times, statistical multiplexing or model parallelism can keep utilization high [Z. Li et al. 2023]. *Routing* sidesteps contention by forwarding each request to the cheapest variant that meets the required SLO [Razavi et al. 2024; Romero et al. 2021]. Both strategies face a fairness problem: without isolation, heavyweight models starve lightweight neighbors of GPU cycles [Sheng et al. 2024]. Parameter-efficient fine-tuning (PEFT) introduces yet another scheduling axis: systems hosting many LoRA adapters must decide when to merge and unmerge adapter weights, a decision that depends on the composition of each incoming batch [B. Wu et al. 2024].

Serverless and autoscaling. Serverless inference eliminates explicit capacity planning but introduces cold-start penalties. Model weights must be loaded into GPU memory before the first request can be served, and that loading time can dwarf the inference itself. Container pre-warming [Y. Yang et al. 2022] and multi-tier checkpoint loading [Fu et al. 2024] shrink these penalties considerably. Still, field reports consistently point to cold starts and

scarce GPU access, not algorithmic limitations, as the primary adoption barriers [Y. Wu et al. 2022]. In practice, the bottleneck has shifted from software design to infrastructure provisioning.

5.4 Pipeline and Governance Techniques

Good operational discipline multiplies every gain achieved at the model and compiler layers; poor discipline can negate them entirely. The individual techniques are not exotic, but sustaining them across the constantly evolving artifacts and hardware of a production stack demands persistent engineering effort.

Regression gates and quality checks. Conventional software often fails loudly when a build breaks; ML systems can degrade silently. A new model version might reduce accuracy by a fraction of a percent, a regression invisible to standard unit tests. Well-designed ML pipelines therefore gate promotion on statistical drift, latency percentiles, and resource consumption relative to a known baseline, not merely on whether the code compiles [Nellutla 2026]. Blueprints for such gates are available, but wiring them into stacks maintained by multiple teams with different toolchains is an engineering project in itself.

Reproducible builds and version pinning. Compiled inference artifacts are tightly coupled to the hardware and software environment that produced them. Pinning model weights is therefore necessary but not sufficient; compiler versions, GPU drivers, and container hashes all need to be locked down, because a mismatch in any one of them can unpredictably alter model behavior (§4.3). CodaBench [Z. Xu et al. 2022] and DLHub [Chard et al. 2019] provide tooling for reproducible evaluations and provenance tracking, though uptake outside benchmark-focused research groups has been slow.

Monitoring and traceability. CPU load, network I/O, and memory utilization tell operators whether the *server* is healthy; they say nothing about whether the *model* is. Catching data drift and accuracy regressions requires model-level proxies, such as prediction-confidence distributions and per-version tail-latency trends, that most monitoring stacks do not collect [Nellutla 2026]. Case studies confirm that embedding monitoring directly into the serving system, rather than bolting it on as a separate layer, improves both throughput and operational safety [Brewer et al. 2021]. Centralizing serving on hardware-specialized infrastructure further simplifies auditing and rollback [Wei et al. 2022].

5.5 What Generalizes, and Why

Reading these four layers together, rather than in isolation, exposes a consistent principle governing which mitigations transfer and which do not. Techniques generalize across workloads and hardware when they target an *invariant* of the deployment, a bottleneck or interface that is present regardless of the specific model or chip, and they fail to transfer when they exploit a *contingent* property of one target. Three patterns recur. First, interventions aimed at a physical resource that every backend shares, such as the memory-bandwidth wall of autoregressive decode (§7) or the end-to-end pipeline stages outside the model (§6), generalize broadly because the resource constraint is universal; KV-cache traffic management and phase-decoupled scheduling are effective on any decoder. Second, interventions expressed above the hardware boundary, at a retargetable IR rather than in hand-tuned kernels, survive migration, whereas empirically autotuned kernels are, by design, welded to the device they were tuned on [Chen et al. 2018; Lattner et al. 2021; Shafi et al. 2021]. Third, statistical and organizational techniques (regression gates, lineage, multiplexing policy) are the most portable of all because they operate on request streams and artifacts, not on tensors, and so are largely indifferent to the underlying silicon. The corollary, which is our interpretation rather than a result established by any single study, is a warning: any reported gain that depends on a specific operator set, precision recipe, or driver version should be read as a lower bound on portability risk, not as a transferable result. This is the lens we apply when interpreting the benchmarking evidence in the next section.

6 Evaluation and Benchmarking Pitfalls

Much of the disagreement in the deployment literature reflects how systems are *measured*, not how they actually differ. Three recurring methodological pitfalls (§6.1–§6.3) account for a disproportionate share of contradictory findings. We survey benchmark coverage in Table 5 and propose a reproducibility checklist (§6.4) designed to prevent them.

6.1 Metric Selection

Academic evaluations default to mean latency and peak throughput, but neither tells a deployment engineer much. Mean latency hides tail behavior: Kasture and Sanchez [2016] showed that two systems with identical means can diverge by an order of magnitude at the 99th percentile, a threshold that often determines whether a user retries or gives up. Peak throughput is equally deceptive when measured on a fully saturated GPU, since production traffic arrives in bursts, not at steady state. A third metric, cost-per-inference, often dominates real deployment decisions [Romero et al. 2021] yet is seldom reported. ***Practitioner takeaway.*** At a minimum, credible deployment benchmarks should include median and tail latencies (p95, p99), throughput under burst traffic, accuracy on drifted data, and monetary or energy cost per inference.

6.2 Hardware and Workload Confounders

Hardware and workload conditions vary across studies, but authors frequently fail to report them, making it impossible to tell whether an improvement is real or an artifact of the test environment. MLPerf Inference [Reddi, Cheng, et al. 2020] partially addresses the problem by standardizing offline and server scenarios; MLPerf Mobile [Reddi, Kanter, et al. 2022] goes further, documenting device-level confounders like thermal throttling and chassis design. Energy consumption introduces yet another axis of variation, shifting with batch size and GPU generation [Jahanshahi et al. 2020]. The risk is not hypothetical. Dacrema et al. [2021] showed that most "state-of-the-art" neural recommendation models published at top venues were outperformed by decade-old baselines once hardware and evaluation conditions were properly controlled. Deployment benchmarks are susceptible to the same inflation.

6.3 End-to-End vs. Component-Level Measurement

Reporting only neural-network execution time while ignoring data loading, preprocessing, and serialization inflates the perceived impact of model-level optimizations. The SeBS benchmark [Copik et al. 2021] measures client, provider, and in-function execution time separately and reports that provider and client overheads are often substantial relative to in-function work—for inference workloads such as warm `image-recognition`, platform latency rivals or exceeds the ResNet forward pass. In serverless settings, halving in-function compute therefore may leave much of the user-visible latency unchanged. ***Practitioner takeaway.*** Evaluations meant to guide deployment decisions should report full pipeline latency, with a stage-by-stage breakdown.

6.4 Reproducibility Checklist

NeurIPS and ICML now require reproducibility checklists for training experiments, but no equivalent standard covers deployment results. ***Practitioner takeaway.*** Building on Olszewski et al. [2023] and CodaBench [Z. Xu et al. 2022], we recommend that teams publishing deployment results report the following:

(1) **Model artifact:** exact weights, architecture definition, and export script.
(2) **Environment:** GPU model, memory, driver version, compiler version, container image.
(3) **Workload:** input shapes, batch sizes, sequence lengths, and traffic model (static vs. trace-based).
(4) **Metrics:** mean, median, p95, p99 latency; hardware costs; warm-up duration.
(5) **Randomness:** fixed seeds and variance across runs.

Table 5. Coverage of existing benchmarks across the five optimization gap dimensions. ✓ = explicitly measured; ~ = partially covered; — = not addressed. No single benchmark spans all five dimensions, motivating the combined benchmark suite proposed in §9.

Benchmark	Perf.	Effic.	Compat.	Observ.	Oper.
MLPerf Inference [Reddi, Cheng, et al. 2020]	✓	~	—	—	—
MLPerf Mobile [Reddi, Kanter, et al. 2022]	✓	✓	~	—	—
TailBench [Kasture and Sanchez 2016]	✓	—	—	—	—
SeBS [Copik et al. 2021]	✓	~	—	—	~
GPU-NEST [Jahanshahi et al. 2020]	~	✓	—	—	—
Codabench [Z. Xu et al. 2022]	~	—	—	—	~

(6) **Optimization settings:** precision (FP32/FP16/INT8), calibration data, and compiler profile boundaries.

7 Cross-Domain Case Studies

Which dimension of the optimization gap bites hardest depends on the workload, and on the hardware it runs on. The four case studies below (vision, LLM, edge, and heterogeneous non-Nvidia hardware) show how different workload characteristics and backends reshuffle the priority order.

7.1 Vision Workloads

The industry's migration from CNNs to Vision Transformers (ViTs) has reopened optimization gaps that many teams assumed were closed. ViT softmax and layer-normalization operations amplify rounding errors under naive PTQ [Liu et al. 2021], and the instance-aware calibration required to tame them (§5) [Moon et al. 2024] invalidates quantization recipes that worked for CNNs, so accuracy must be re-validated per target platform. Cloud-versus-edge benchmarks sharpen the picture: datacenter compilers win on throughput but burn energy at rates no battery-powered device can sustain, while lighter runtimes trade speed for efficiency [Koubaa et al. 2022]. For real-time vision, quantization and scheduling must be co-designed around the thermal envelope of the target hardware [Pappula 2023].

7.2 NLP and LLM Workloads

With LLMs, the bottleneck migrates from compute to memory bandwidth. Compressing the model matters less than managing the memory it consumes at runtime.

Speculative decoding. A lightweight "draft" model proposes several tokens at once; the full model then verifies them in a single forward pass, preserving output quality exactly [Spector and Ré 2023; Svirschevski et al. 2024; H. Xia et al. 2024]. The technique works well when the draft model's token distribution closely tracks the target's. When alignment is poor, most proposals are rejected and speculative decoding degenerates into standard autoregressive generation with added overhead. Recent work extends the idea to multimodal inputs and massively parallel interactive settings [Gagrani et al. 2024; Svirschevski et al. 2024].

KV cache management. Because the KV cache dominates memory as sequences lengthen (§4.2), the highest-leverage LLM interventions target memory *traffic* rather than model size: restructured attention reads [Sanovar et al. 2025], prefetching into on-chip SRAM [Yüzügüler et al. 2026], and selective eviction policies. These generalize well across models precisely because they attack a bandwidth bottleneck common to all autoregressive decoders, whereas an equivalent effort spent on further weight compression yields diminishing returns [Sanovar et al. 2025; Shi et al. 2024].

Table 6. Comparison of inference and LLM serving systems surveyed. Each system targets a different bottleneck in the serving pipeline; reported gains are as stated by the original authors.

System	Key Mechanism	Target Problem	Reported Gain
AlpaServe [Z. Li et al. 2023]	Statistical multiplexing + model parallelism	Multi-model throughput	10× higher request rate
Sarathi-Serve [A. Agrawal et al. 2024]	Prefill/decode decoupling	Throughput-latency trade-off	2.6× capacity (Mistral-7B)
Llumnix [Sun et al. 2024]	Live request migration	Tail latency under load	10× tail-latency reduction
dLoRA [B. Wu et al. 2024]	Merged/unmerged LoRA switching	Multi-adapter serving	57.9× over vLLM
ServerlessLLM [Fu et al. 2024]	Multi-tier checkpoint loading	Cold-start latency	10–200× faster startup
INFaaS [Romero et al. 2021]	Automated variant selection	Cost/SLO optimization	1.3× throughput, 1.6× fewer violations
Sponge [Razavi et al. 2024]	In-place vertical scaling	Per-request SLO adjustment	Dynamic GPU reallocation
INFless [Y. Yang et al. 2022]	Pre-warming + co-location	Serverless inference	2–5× throughput

Continuous serving. Beyond the prefill/decode decoupling covered in §5, a newer challenge is emerging: as organizations deploy dozens of LoRA adapters per base model, adapter-weight management becomes a scheduling problem in its own right [B. Wu et al. 2024], a layer of complexity absent from single-model deployments and one that resists the single-bottleneck assumption most serving benchmarks still make.

7.3 Edge Deployment Workloads

Edge hardware flips the priority order: thermal management and battery life often matter more than raw throughput. Datacenter compiler profiles perform poorly on edge chips with heterogeneous memory hierarchies and dynamic voltage and frequency scaling (DVFS) constraints. Frequency policies and model precision must be co-optimized for the target silicon [Jeong et al. 2022; Marco et al. 2020; Shafi et al. 2021; Wang et al. 2021]. Enterprise edge-AI adoption is accelerating [Almeida et al. 2021; Manzoor et al. 2022], and with it comes pressure to run multiple models concurrently on resource-limited chips. At that point, system-level orchestration, not model compression, becomes the primary barrier [Sander et al. 2025; Seo et al. 2021; Shankar 2024; W. Xiao et al. 2024; Z. Yang et al. 2021; Z. Zhang et al. 2023]. The parallel with LLM serving is hard to miss: once several models share constrained hardware, the limiting factor shifts from shrinking individual models to orchestrating the fleet.

7.4 Heterogeneous Hardware Ecosystems

The published deployment literature, and consequently much of this survey, tilts toward Nvidia GPUs and CUDA-based tooling. Yet production inference increasingly runs on a heterogeneous fleet, and the optimization gap manifests differently on each backend. Reading the taxonomy (§4) across hardware classes is instructive precisely because the *binding dimension* shifts with the architecture, not just with the workload.

Data-center accelerators: TPUs and AMD GPUs. Google TPUs illustrate a different point on the design curve: a systolic-array datapath tuned for dense matrix multiplication delivers high throughput per watt [Jouppi, Kurian, et al. 2023; Jouppi, Young, et al. 2017]. The flip side of that specialization is a narrower fast path, so the *compatibility* dimension tends to bind first: an operation outside the well-supported set falls back to a slower path or must be re-expressed before it will lower efficiently. AMD data-center GPUs face a portability gap of a different kind: the hardware is competitive, but the surrounding software ecosystem (ROCm and its compiler backends) is widely reported to trail CUDA in kernel coverage and tooling maturity [Chen et al. 2018; M. Li et al. 2021], so kernels autotuned on Nvidia hardware typically must be re-derived rather than simply ported. In both cases the lesson from §5 holds: gains anchored to a vendor-neutral IR (via TVM or MLIR) transfer, whereas gains anchored to vendor-specific kernels do not [Chen et al. 2018; Lattner et al. 2021].

NPUs, FPGAs, and CPUs. Mobile and embedded NPUs push the *efficiency* dimension to the fore: fixed-function datapaths achieve excellent energy-per-inference but expose severe compatibility cliffs, since unsupported operators must run on a companion CPU or GPU, fragmenting the graph. FPGAs trade generality for the ability to co-design the datapath with the model; frameworks such as FINN synthesize custom, heavily quantized dataflow architectures that reach very low latency, at the cost of a lengthy synthesis-and-validation cycle that makes the *operational* dimension (rebuild and re-verification on every model change) the dominant constraint [Umuroglu et al. 2017]. CPUs remain the default fallback for tail workloads and cost-sensitive services; asymmetric multicore designs make scheduling and clock policy, rather than raw compute, the lever that matters [Wang et al. 2021], and INT8 quantization recipes that are safe on one accelerator can shift accuracy differently on another [Gholami et al. 2022; Rachmanto et al. 2024].

Custom cloud silicon and dataflow accelerators. Hyperscalers and start-ups increasingly build their own inference silicon, and here the binding dimension is usually *compatibility* in its software sense: the hardware is only as fast as the vendor's compiler stack lets a model be. Meta's MTIA, an ASIC co-designed for deep learning recommendation models and integrated with PyTorch, averaged about 0.9× the performance per watt of a GPU across production recommendation models while reaching up to 2× on GEMM-dominated operators; the authors attribute much of the remaining gap to the maturity of the emerging software stack rather than to the silicon [Firoozshahian et al. 2023]. A cross-vendor evaluation of dataflow accelerators from SambaNova, Cerebras, Graphcore, and Groq likewise found that each ships its own programming model and software stack, so porting a workload and comparing results fairly is itself a substantial engineering effort [Emani et al. 2022]. Other hyperscaler designs, such as AWS Inferentia and Trainium and Microsoft's Maia, are documented mainly in vendor materials rather than in peer-reviewed measurement studies, which is itself a sign of how thin independent evidence on custom cloud silicon remains (§9.1).

Synthesis. Across these backends a single generalization holds: no optimization is portable in the abstract; it is portable only with respect to the dimension a given architecture stresses. A quantization recipe that is an efficiency win on a GPU can be a compatibility hazard on an NPU; an autotuned kernel that is a performance win on one vendor is dead weight on another. ***Practitioner takeaway.*** Before committing to a backend, identify which taxonomy dimension that hardware class stresses first (Table 2), and prefer optimizations expressed at a hardware-neutral IR so that effort survives the next hardware migration.

8 Security, Privacy, Robustness, and Compliance

Deployment introduces failure modes that never appear in offline evaluation because they arise only once a model is exposed to real users, real data, and adversaries. These are facets of the operational-robustness dimension (§4): what fails is the process and interface around the model, not its offline accuracy. We treat them together here because the mitigations interact with, and sometimes conflict with, the performance and efficiency optimizations discussed earlier.

Inference-time attack surface. A served model exposes two exploitable channels. Prediction APIs enable *model extraction*, in which an adversary reconstructs a close functional copy of a proprietary model from query–response pairs [Tramèr, F. Zhang, et al. 2016], eroding the very cost advantage that motivated optimization. Confidence scores enable *membership inference*, revealing whether a particular record was in the training data, a direct privacy violation with regulatory consequences [Shokri et al. 2017]. Rate limiting, output coarsening, and confidence-masking mitigate both, but each trades away some of the observability and accuracy that the serving layer otherwise provides.

Robustness under optimization. Adversarial perturbations that flip predictions [Goodfellow et al. 2015] interact with deployment optimizations in underappreciated ways. Quantization and operator fusion can alter a model's decision boundary, and empirical work has observed that compilation can incidentally change behavior on corrupted or adversarial inputs, with no guarantee of the direction in advance [Shafi et al. 2021]. The implication is that robustness under distribution shift and attack, like accuracy, must be re-validated after every optimization pass rather than assumed to carry over from the reference model.

Privacy-preserving and confidential inference. Where data sensitivity is paramount, differentially private *training* bounds how much any single example can influence the model and thereby limits membership-inference risk at inference time, though the privacy noise budget competes with accuracy [Abadi et al. 2016; Shokri et al. 2017]; trusted-hardware execution (e.g., running inference inside a hardware enclave) protects weights and inputs from an untrusted host at a throughput cost [Tramèr and Boneh 2019]. Both mechanisms reintroduce the efficiency and performance trade-offs of §4, so they cannot be bolted on without re-tuning the deployment.

Compliance and lineage. Regulated domains increasingly require that every production prediction be attributable to a specific model version, dataset, and configuration. The transparent variant-switching that improves cost-efficiency (§5) directly undermines this lineage, creating a genuine tension between operational efficiency and auditability. ***Practitioner takeaway.*** Treat security, privacy, and compliance as first-class deployment requirements with their own regression gates and lineage logging, and re-measure accuracy and behavior under distribution shift and attack after each optimization pass; retrofitting these controls after an incident is far costlier than budgeting for them up front.

9 Best Practices, Open Problems, and Research Directions

Drawing on the literature surveyed, we distill practical guidelines for each deployment phase; consistent with our convention (§1), the guidelines below are ***practitioner guidance*** rather than a summary of findings. *Before deployment*: (1) lock down SLOs early, because tail-latency guarantees drive fundamentally different architecture choices than throughput targets [Kasture and Sanchez 2016]; (2) numerically verify every exported model against its training-framework counterpart [Jajal et al. 2024]; (3) profile on the actual target hardware, not a proxy [Lechner and Jantsch 2021; Shafi et al. 2021]; (4) benchmark end-to-end, including data loading and network transit [Copik et al. 2021]; (5) when more than one hardware target is plausible, express optimizations at a hardware-neutral IR so that effort survives migration (§7.4) [Chen et al. 2018; Lattner et al. 2021]. *After deployment*: (1) automate CI/CD gates covering accuracy, latency, and memory regression [Nellutla 2026]; (2) pin every dependency, including compiler versions, GPU drivers, and container hashes [Shafi et al. 2021]; (3) alert on p99 latency and log proxy metrics per request [Nellutla 2026]; (4) maintain prediction lineage for auditing and rollback [Wei et al. 2022]; (5) guard the inference interface against extraction and membership-inference attacks, and re-validate accuracy and robustness under distribution shift after each optimization pass (§8) [Shokri et al. 2017; Tramèr, F. Zhang, et al. 2016].

Five research challenges remain open, each cutting across multiple stack layers. The gaps themselves are documented in the literature cited; the directions we propose for closing them are our own assessment.

Automating optimization policies. Choosing precision, sparsity ratio, batch size, and server allocation is a combinatorial problem that practitioners currently solve through manual experimentation and hard-won intuition. Dynamic per-layer precision [Shinde 2024], weight-sharing NAS methods that address super-network/sub-architecture instability [L. Xie et al. 2022], and automated routing [Ahmad et al. 2024; Romero et al. 2021] address individual subproblems, but no framework jointly optimizes model, compiler, and serving configuration. A major reason is that these choices interact nonlinearly, which makes exhaustive search intractable.

Cross-hardware portability. Compilation gains remain architecture-specific [Lindkvist 2024], and ONNX's promise of universal exchange is undermined by converter bugs [Jajal et al. 2024]. Retargetable IRs such as TVM and MLIR are partial answers [Chen et al. 2018; Lattner et al. 2021], but their coverage across the full spread of production silicon (TPUs, AMD GPUs, mobile NPUs, FPGAs, and custom ASICs) is uneven, and each new backend reopens the operator-coverage problem. The field needs portable performance abstractions, that is, intermediate representations that expose hardware-specific optimizations without forcing per-target recompilation, together with automated conformance testing to verify numerical correctness across the full range of hardware targets rather than a single dominant vendor.

Deployment-time security and compliance. Security, privacy, and auditability remain afterthoughts in most deployment tooling (§8). No widely used framework jointly measures a model's exposure to extraction and membership-inference attacks, the robustness drift induced by quantization and fusion, and the lineage guarantees required for regulatory audit. Building these checks into the same CI/CD and benchmarking machinery that already gates accuracy and latency, so that a model cannot ship unless it also clears a security and compliance bar, is an open and increasingly urgent problem.

Observability for compiled graphs. Today's profilers pinpoint slow kernels but cannot attribute them to the original model layers they came from. Bridging that gap requires new frameworks that map hardware metrics back to model-level semantics. This is especially pressing for LLMs, whose prefill and decode phases exhibit qualitatively different compute profiles [Sanovar et al. 2025].

Unified deployment benchmarks. MLPerf [Reddi, Cheng, et al. 2020] standardizes speed measurements, but no widely adopted benchmark jointly evaluates quantization-induced accuracy loss, cold-start behavior, rollback resilience, and energy consumption, which together define the full spectrum of deployment concerns. Merging MLPerf with serverless benchmarks like SeBS [Copik et al. 2021] into a single suite would, for the first time, cover all five taxonomy dimensions in one evaluation.

9.1 Limitations

We state the boundaries of this review explicitly so that readers can calibrate its conclusions. *Sources.* Our analysis draws exclusively on publicly available research; proprietary serving stacks are largely undocumented and therefore absent, which likely understates the maturity of operational and security practice inside large providers. *Hardware coverage.* Although we deliberately cover TPUs, AMD GPUs, NPUs, FPGAs, CPUs, and custom cloud silicon (§7.4), the underlying corpus still tilts toward the Nvidia ecosystem because published, quantitative deployment work overwhelmingly targets Nvidia hardware; evidence on non-Nvidia accelerators, and especially on cross-vendor comparisons under identical workloads, remains comparatively thin (for hyperscaler ASICs such as AWS Inferentia and Trainium or Microsoft Maia, peer-reviewed measurements are scarce), and our claims about those platforms are correspondingly more tentative. *Intentional exclusions.* We do not cover training-time systems, hardware design and chip architecture, or federated and on-device *learning*; we touch privacy and security only as deployment concerns (§8) rather than as fields in their own right. *Reproduction.* We report speedup and accuracy figures as stated by the original authors, without independent reproduction, and inherit whatever measurement bias those studies carry. *Source quality.* A small number of cited works are preprints, theses, or industry-adjacent surveys (e.g., MLOps reference architectures and TensorRT optimization studies); we rely on them only where peer-reviewed alternatives were unavailable and do not treat them as primary evidence for quantitative claims. *Shelf life.* Several areas are moving fast enough that specific numbers will date quickly: LLM continuous batching, KV-cache management, speculative decoding, and serverless cold-start mitigation are all under active, rapid development, so their quantitative results reflect early-2026 conditions even though the

structural tensions they expose, and the taxonomy that organizes them, are not tied to any particular hardware generation or serving framework.

10 Conclusion

Bridging the gap between benchmark accuracy and reliable production operation remains one of applied machine learning's most stubborn problems.

Several lessons stand out. First, no single layer of the stack is sufficient on its own. A well-executed quantization strategy cannot prevent latency spikes if the serving system lacks dynamic batching; compiler gains evaporate if the CI/CD pipeline fails to catch regressions. Deployment quality is set by the *weakest link* across the stack, not by the strongest.

Second, the binding constraint depends on the workload. LLMs are memory-bandwidth-bound and benefit most from speculative decoding and KV-cache management. Edge vision models hit thermal and battery ceilings long before compute becomes scarce. Multi-tenant cloud deployments are limited primarily by interference and scheduling. Any deployment framework that assumes a single dominant bottleneck will systematically underserve at least one of these regimes.

Third, and perhaps most consequentially, observability still lags behind other capabilities. When compiled models are treated as opaque artifacts and reproducibility checks are skipped, the same production failures recur with root causes that remain undiagnosed for weeks. Restoring visibility into compiled graphs is likely the highest-leverage open problem in this space.

The taxonomy presented here offers practitioners a diagnostic tool: identify which constraint binds a given deployment, then select the intervention most likely to relieve it. Standardizing what deployment evaluations report, along the lines of the checklist in §6.4, would let the field compare results across studies and accumulate reliable evidence on which interventions transfer beyond the hardware and workload where they were first demonstrated.

Acknowledgments

We thank the anonymous reviewers and the associate editor for feedback that substantially improved this survey. *Competing interests.* L. Dmello is affiliated with Nvidia Corporation. This survey draws solely on publicly available literature, and we have made a deliberate effort to cover non-Nvidia hardware ecosystems even-handedly (§7.4).

References

M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. 2016. "Deep Learning with Differential Privacy." In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. doi:10.1145/2976749.2978318.

A. Agrawal, N. Kedia, A. Panwar, J. Mohan, N. Kwatra, B. S. Gulavani, A. Tumanov, and R. Ramjee. 2024. "Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve." In: *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

S. Ahmad, H. Guan, B. D. Friedman, T. Williams, R. K. Sitaraman, and T. Woo. 2024. "Proteus: A High-Throughput Inference-Serving System with Accuracy Scaling." In: *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

M. Almeida, S. Laskaridis, A. Mehrotra, L. Dudziak, I. Leontiadis, and N. D. Lane. 2021. "Smart at What Cost? Characterising Mobile Deep Neural Networks in the Wild." In: *ACM Internet Measurement Conference (IMC)*.

D. S. Berger, B. Berg, T. Zhu, S. Sen, and M. Harchol-Balter. 2018. "RobinHood: Tail Latency Aware Caching—Dynamic Reallocation from Cache-Rich to Cache-Poor." In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

K. Bhardwaj, C. Lin, A. L. Sartor, and R. Marculescu. 2020. "Memory- and Communication-Aware Model Compression for Distributed Deep Learning Inference on IoT." *ACM Transactions on Internet of Things*. doi:10.1145/3358205.

M. Boehm, B. Reinwald, D. Hutchison, P. Sen, A. V. Evfimievski, and N. Pansare. 2018. "On Optimizing Operator Fusion Plans for Large-Scale Machine Learning in SystemML." *Proceedings of the VLDB Endowment*, 11, 12, 1755–1768. doi:10.14778/3229863.3229865.

Y. Bondarenko, R. D. Chiaro, and M. Nagel. 2024. "Low-Rank Quantization-Aware Training for LLMs." In: *ICML 2024 Workshop on Efficient Systems for Foundation Models (ES-FoMo II)*.

W. Brewer, D. Martinez, M. Boyer, D. Jude, A. Wissink, B. Parsons, J. Yin, and V. Anantharaj. 2021. "Production Deployment of Machine-Learned Rotorcraft Surrogate Models on HPC." In: *IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (ML-HPC)*. doi:10.1109/MLHPC54614.2021.00008.

R. Chard, L. Ward, Z. Li, Y. Babuji, A. Woodard, S. Tuecke, K. Chard, B. Blaiszik, and I. Foster. 2019. "DLHub: Model and Data Serving for Science." In: *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*.

T. Chen et al. 2018. "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning." In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

M. Copik, G. Kwaśniewski, M. Besta, M. Podstawski, and T. Hoefler. 2021. "SeBS: A Serverless Benchmark Suite for Function-as-a-Service Computing." In: *22nd International Middleware Conference (Middleware)*.

M. F. Dacrema, S. Boglio, P. Cremonesi, and D. Jannach. 2021. "A Troubling Analysis of Reproducibility and Progress in Recommender Systems Research." *ACM Transactions on Information Systems*, 39, 2. doi:10.1145/3434185.

M. Emani et al. 2022. "A Comprehensive Evaluation of Novel AI Accelerators for Deep Learning Workloads." In: *IEEE/ACM International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, 13–25. doi:10.1109/PMBS56514.2022.00007.

A. Firoozshahian, J. Coburn, R. Levenstein, R. Nattoji, A. Kamath, O. Wu, et al. 2023. "MTIA: First Generation Silicon Targeting Meta's Recommendation Systems." In: *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA)*. doi:10.1145/3579371.3589348.

Y. Fu, L. Xue, Y. Huang, A.-O. Brabete, D. Ustiugov, Y. Patel, and L. Mai. 2024. "ServerlessLLM: Low-Latency Serverless Inference for Large Language Models." In: *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

M. Gagrani, R. Goel, W. Jeon, J. Park, M. Lee, and C. Lott. 2024. "On Speculative Decoding for Multimodal Large Language Models." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. ELVM Workshop. https://arxiv.org/abs/2404.08856.

H. Gao, B. Qiu, Y. Wang, S. Yu, Y. Xu, and X. Wang. 2024. "TBDB: Token Bucket-Based Dynamic Batching for Resource Scheduling Supporting Neural Network Inference in Intelligent Consumer Electronics." *IEEE Transactions on Consumer Electronics*, 70, 1, 1134–1144. doi:10.1109/TCE.2023.3339633.

A. K. A. A. Ghadani, W. Mateen, and R. G. Ramaswamy. 2020. "Tensor-Based CUDA Optimization for ANN Inferencing Using Parallel Acceleration on Embedded GPU." In: *Artificial Intelligence Applications and Innovations (AIAI), IFIP Advances in Information and Communication Technology*. Springer. doi:10.1007/978-3-030-49161-1_25.

A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer. 2022. "A Survey of Quantization Methods for Efficient Neural Network Inference." In: *Low-Power Computer Vision: Improve the Efficiency of Artificial Intelligence*. arXiv:2103.13630. Chapman and Hall/CRC.

I. J. Goodfellow, J. Shlens, and C. Szegedy. 2015. "Explaining and Harnessing Adversarial Examples." In: *International Conference on Learning Representations (ICLR)*.

M. Haroush, I. Hubara, E. Hoffer, and D. Soudry. 2020. "The Knowledge Within: Methods for Data-Free Model Compression." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 8494–8502. doi:10.1109/CVPR42600.2020.00852.

I. Hubara, Y. Nahshan, Y. Hanani, R. Banner, and D. Soudry. 2021. "Accurate Post Training Quantization With Small Calibration Sets." In: *International Conference on Machine Learning (ICML)*.

A. Jahanshahi, H. Z. Sabzi, C. Lau, and D. Wong. 2020. "GPU-NEST: Characterizing Energy Efficiency of Multi-GPU Inference Servers." *IEEE Access*, 8, 153756–153769. doi:10.1109/ACCESS.2020.3018170.

P. Jajal, W. Jiang, A. Tewari, E. Kocinare, J. Woo, A. Sarraf, Y.-H. Lu, G. K. Thiruvathukal, and J. C. Davis. 2024. "Interoperability in Deep Learning: A User Survey and Failure Analysis of ONNX Model Converters." In: *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*.

E. Jeong, J. Kim, and S. Ha. 2022. "TensorRT-Based Framework and Optimization Methodology for Deep Learning Inference on Jetson Boards." *ACM Transactions on Embedded Computing Systems*, 21, 5. doi:10.1145/3508391.

N. P. Jouppi, G. Kurian, et al. 2023. "TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings." In: *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA)*. doi:10.1145/3579371.3589350.

N. P. Jouppi, C. Young, et al. 2017. "In-Datacenter Performance Analysis of a Tensor Processing Unit." In: *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*. doi:10.1145/3079856.3080246.

H. Kasture and D. Sanchez. 2016. "TailBench: A Benchmark Suite and Evaluation Methodology for Latency-Critical Applications." In: *IEEE International Symposium on Workload Characterization (IISWC)*. doi:10.1109/IISWC.2016.7581261.

A. Koubaa, A. Ammar, A. Kanhouch, and Y. AlHabashi. 2022. "Cloud Versus Edge Deployment Strategies of Real-Time Face Recognition Inference." *IEEE Access*.

C. Lattner et al. 2021. "MLIR: Scaling Compiler Infrastructure for Domain Specific Computation." In: *Proceedings of the IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. doi:10.1109/CGO51591.2021.9370308.

M. Lechner and A. Jantsch. 2021. "Blackthorn: Latency Estimation Framework for CNNs on Embedded NVIDIA Platforms." *IEEE Access*. doi:10.1109/ACCESS.2021.3101936.

B. Li, Y. Jiang, V. Gadepally, and D. Tiwari. 2024. "LLM Inference Serving: Survey of Recent Advances and Opportunities." In: *IEEE High Performance Extreme Computing Conference (HPEC)*. doi:10.1109/HPEC62836.2024.10938426.

H. Li, F. Liu, Z. Sun, Z. Wang, S. Huang, N. Yang, and L. Jiang. 2025. "NeuronQuant: Accurate and Efficient Post-Training Quantization for Spiking Neural Networks." In: *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*.

M. Li et al. 2021. "The Deep Learning Compiler: A Comprehensive Survey." *IEEE Transactions on Parallel and Distributed Systems*, 32, 3, 708–727. doi:10.1109/TPDS.2020.3030548.

Z. Li et al. 2023. "AlpaServe: Statistical Multiplexing with Model Parallelism for Deep Learning Serving." In: *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

A. Lindkvist. 2024. *Optimization of Machine Learning Inference with TensorRT*. Bachelor's thesis, Karlstad University. (2024).

Z. Liu, Y. Wang, K. Han, W. Zhang, S. Ma, and W. Gao. 2021. "Post-Training Quantization for Vision Transformer." In: *Advances in Neural Information Processing Systems (NeurIPS)*.

S. Manzoor, E.-J. Kim, S.-H. Joo, S.-H. Bae, G.-G. In, K.-J. Joo, J.-H. Choi, and T.-Y. Kuc. 2022. "Edge Deployment Framework of GuardBot for Optimized Face Mask Recognition With Real-Time Inference Using Deep Learning." *IEEE Access*. doi:10.1109/ACCESS.2022.3190538.

V. S. Marco, B. Taylor, Z. Wang, and Y. Elkhatib. 2020. "Optimizing Deep Learning Inference on Embedded Systems Through Adaptive Model Selection." *ACM Transactions on Embedded Computing Systems*, 19, 1. doi:10.1145/3371154.

J. Moon, D. Kim, J. Cheon, and B. Ham. 2024. "Instance-Aware Group Quantization for Vision Transformers." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. https://arxiv.org/abs/2404.00928.

A. Musa, H. A. Kakudi, M. Hassan, M. Hamada, U. Umar, and M. L. Salisu. 2025. "Lightweight Deep Learning Models for Edge Devices—A Survey." *International Journal of Computer Information Systems and Industrial Management Applications*, 17, 189–206.

N. Nellutla. 2026. "MLOps 2.0: A Reference Architecture for CI/CD with Always-On Data Quality Gates." *International Journal of Artificial Intelligence and Machine Learning*, 6, 1, 118–131. doi:10.51483/IJAIML.6.1.2026.118-131.

W. Niu, J. Guan, Y. Wang, G. Agrawal, and B. Ren. 2021. "DNNFusion: Accelerating Deep Neural Networks Execution with Advanced Operator Fusion." In: *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI)*. doi:10.1145/3453483.3454083.

D. Olszewski, A. Lu, C. Stillman, K. Warren, C. Kitroser, A. Pascual, P. Traynor, and K. Butler. 2023. ""Get in Researchers; We're Measuring Reproducibility": A Reproducibility Study of Machine Learning Papers in Tier 1 Security Conferences." In: *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM. doi:10.1145/3576915.3623130.

K. K. Pappula. 2023. "Edge-Deployed Computer Vision for Real-Time Defect Detection." *International Journal of AI, BigData, Computational and Management Studies*, 4, 3, 72–81. doi:10.63282/3050-9416.IJAIBDCMS-V4I3P108.

R. D. Rachmanto, Z. Sukma, A. N. L. Nabhaan, A. Setyanto, T. Jiang, and I. K. Kim. 2024. "Characterizing Deep Learning Model Compression with Post-Training Quantization on Accelerated Edge Devices." In: *IEEE International Conference on Edge Computing (EDGE)*. doi:10.1109/EDGE62653.2024.00023.

K. Razavi, S. Ghafouri, M. Mühlhäuser, P. Jamshidi, and L. Wang. 2024. "Sponge: Inference Serving with Dynamic SLOs Using In-Place Vertical Scaling." In: *EuroMLSys*.

V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, et al. 2020. "MLPerf Inference Benchmark." In: *ACM/IEEE International Symposium on Computer Architecture (ISCA)*.

V. J. Reddi, D. Kanter, et al. 2022. "MLPerf Mobile Inference Benchmark." In: *Proceedings of Machine Learning and Systems (MLSys)*.

F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis. 2021. "INFaaS: Automated Model-less Inference Serving." In: *2021 USENIX Annual Technical Conference (USENIX ATC)*. USENIX Association, 397–411.

M. Salmani, S. Ghafouri, A. Sanaee, K. Razavi, M. Mühlhäuser, J. Doyle, P. Jamshidi, and M. Sharifi. 2023. "Reconciling High Accuracy, Cost-Efficiency, and Low Latency of Inference Serving Systems." In: *EuroMLSys*.

S. D. Samarasinghe. 2025. "Evaluating Dynamic Batching Strategies for Energy-Efficient Inference Serving: A Performance Study." Master's thesis. Aalto University.

J. Sander, A. Cohen, V. R. Dasari, B. Venable, and B. Jalaian. 2025. *On Accelerating Edge AI: Optimizing Resource-Constrained Environments*. arXiv preprint arXiv:2501.15014. (2025).

R. Sanovar, S. Bharadwaj, R. S. Amant, V. Rühle, and S. Rajmohan. 2025. "LeanAttention: Hardware-Aware Scalable Attention Mechanism for the Decode-Phase of Transformers." In: *Proceedings of Machine Learning and Systems (MLSys)*.

W. Seo, S. Cha, Y. Kim, J. Huh, and J. Park. 2021. "SLO-Aware Inference Scheduler for Heterogeneous Processors in Edge Platforms." *ACM Transactions on Architecture and Code Optimization*, 18, 4. doi:10.1145/3460352.

O. Shafi, C. Rai, G. Ananthanarayanan, and R. Sen. 2021. "Demystifying TensorRT: Characterizing Neural Network Inference Engine on NVIDIA Edge Devices." In: *IEEE International Symposium on Workload Characterization (IISWC)*. doi:10.1109/IISWC53511.2021.00031.

V. Shankar. 2024. "Edge AI: A Comprehensive Survey of Technologies, Applications, and Challenges." In: *IEEE International Conference on Advances in Computing, Engineering and Technology (ACET)*. doi:10.1109/ACET61898.2024.10730112.

Y. Sheng, S. Cao, D. Li, B. Zhu, Z. Li, D. Zhuo, J. E. Gonzalez, and I. Stoica. 2024. "Fairness in Serving Large Language Models." In: *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 965–988.

L. Shi, H. Zhang, Y. Yao, Z. Li, and H. Zhao. 2024. "Keep the Cost Down: A Review on Methods to Optimize LLM's KV-Cache Consumption." In: *First Conference on Language Modeling (COLM)*.

T. Shinde. 2024. "Adaptive Quantization and Pruning of Deep Neural Networks via Layer Importance Estimation." In: *Workshop on Machine Learning and Compression, Advances in Neural Information Processing Systems (NeurIPS)*.

R. Shokri, M. Stronati, C. Song, and V. Shmatikov. 2017. "Membership Inference Attacks Against Machine Learning Models." In: *IEEE Symposium on Security and Privacy (SP)*. doi:10.1109/SP.2017.41.

B. Spector and C. Ré. 2023. "Accelerating LLM Inference with Staged Speculative Decoding." In: *ICML 2023 Workshop on Efficient Systems for Foundation Models (ES-FoMo)*.

B. Sun, Z. Huang, H. Zhao, W. Xiao, X. Zhang, Y. Li, and W. Lin. 2024. "Llumnix: Dynamic Scheduling for Large Language Model Serving." In: *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

R. Svirschevski, A. May, Z. Chen, B. Chen, Z. Jia, and M. Ryabinin. 2024. "SpecExec: Massively Parallel Speculative Decoding for Interactive LLM Inference on Consumer Devices." In: *Advances in Neural Information Processing Systems (NeurIPS)*.

S. A. Tailor, J. Fernandez-Marques, and N. D. Lane. 2021. "DEGREE-QUANT: Quantization-Aware Training for Graph Neural Networks." In: *Proceedings of the International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=NSBrFgJAHg.

T. Tornede, A. Tornede, J. Hanselle, F. Mohr, M. Wever, and E. Hüllermeier. 2023. "Towards Green Automated Machine Learning: Status Quo and Future Directions." *Journal of Artificial Intelligence Research*.

F. Tramèr and D. Boneh. 2019. "Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware." In: *International Conference on Learning Representations (ICLR)*.

F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. 2016. "Stealing Machine Learning Models via Prediction APIs." In: *25th USENIX Security Symposium (USENIX Security)*.

Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers. 2017. "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference." In: *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. doi:10.1145/3020078.3021744.

G. Verma, Y. Gupta, A. M. Malik, and B. Chapman. 2021. "Performance Evaluation of Deep Learning Compilers for Edge Inference." In: *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 858–865. doi:10.1109/IPDPSW52791.2021.00128.

M. Wang, S. Ding, T. Cao, Y. Liu, and F. Xu. 2021. "AsyMo: Scalable and Efficient Deep Learning Inference on Asymmetric Mobile CPUs." In: *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*.

Y. Wei, M. Langer, F. Yu, M. Lee, K. Liu, J. Shi, and J. Wang. 2022. "A GPU-Specialized Inference Parameter Server for Large-Scale Deep Recommendation Models." In: *Proceedings of the 16th ACM Conference on Recommender Systems (RecSys)*. NVIDIA HugeCTR Hierarchical Parameter Server. ACM.

C. Wohlin. 2014. "Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering." In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE)*. ACM. doi:10.1145/2601248.2601268.

B. Wu, R. Zhu, Z. Zhang, P. Sun, X. Liu, and X. Jin. 2024. "dLoRA: Dynamically Orchestrating Requests and Adapters for LoRA LLM Serving." In: *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius. 2020. "Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation." *arXiv preprint arXiv:2004.09602*.

Y. Wu, T. T. A. Dinh, G. Hu, M. Zhang, Y. M. Chee, and B. C. Ooi. 2022. "Serverless Data Science—Are We There Yet? A Case Study of Model Serving." In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*.

C. Xia, J. Zhao, Q. Sun, Z. Wang, Y. Wen, X. Feng, and H. Cui. 2024. "Optimizing Deep Learning Inference via Global Analysis and Tensor Expressions." In: *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

H. Xia, Z. Yang, Q. Dong, P. Wang, Y. Li, T. Ge, T. Liu, W. Li, and Z. Sui. 2024. "Unlocking Efficiency in Large Language Model Inference: A Comprehensive Survey of Speculative Decoding." In: *Findings of the Association for Computational Linguistics (ACL)*.

D. Xiao, Z. Liu, Y. Yuan, Q. Pang, and S. Wang. 2022. "Metamorphic Testing of Deep Learning Compilers." *Proceedings of the ACM on Measurement and Analysis of Computing Systems*. doi:10.1145/3508035.

W. Xiao, Y. Hao, J. Liang, L. Hu, S. A. Alqahtani, and M. Chen. 2024. "Adaptive Compression Offloading and Resource Allocation for Edge Vision Computing." *IEEE Transactions on Cognitive Communications and Networking*, 10, 6.

L. Xie et al. 2022. "Weight-Sharing Neural Architecture Search: A Battle to Shrink the Optimization Gap." *ACM Computing Surveys*, 54, 9. doi:10.1145/3473330.

F. Xu, J. Xu, J. Chen, L. Chen, R. Shang, Z. Zhou, and F. Liu. 2023. "iGniter: Interference-Aware GPU Resource Provisioning for Predictable DNN Inference in the Cloud." *IEEE Transactions on Parallel and Distributed Systems*, 34, 3.

Z. Xu, S. Escalera, A. Pavão, M. Richard, W.-W. Tu, Q. Yao, H. Zhao, and I. Guyon. 2022. "Codabench: Flexible, easy-to-use, and reproducible meta-benchmark platform." *Patterns*, 3, 7, 100543. doi:10.1016/j.patter.2022.100543.

Y. Yang, L. Zhao, Y. Li, H. Zhang, J. Li, M. Zhao, X. Chen, and K. Li. 2022. "INFless: A Native Serverless System for Low-Latency, High-Throughput Inference." In: *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

Z. Yang, K. Nahrstedt, H. Guo, and Q. Zhou. 2021. "DeepRT: A Soft Real Time Scheduler for Computer Vision Applications on the Edge." In: *ACM/IEEE Symposium on Edge Computing (SEC)*.

Z. Yao, R. Y. Aminabadi, M. Zhang, X. Wu, C. Li, and Y. He. 2022. "ZeroQuant: Efficient and Affordable Post-Training Quantization for Large-Scale Transformers." In: *Advances in Neural Information Processing Systems (NeurIPS)*.

F. Yu, D. Wang, L. Shangguan, M. Zhang, C. Liu, and X. Chen. 2022. "A Survey of Multi-Tenant Deep Learning Inference on GPU." In: *MLSys 2022 Workshop on Cloud Intelligence / AIOps*. arXiv:2203.09040.

A. C. Yüzügüler, J. Zhuang, and L. Cavigelli. 2026. "Brief Announcement: PRESERVE: Prefetching Model Weights and KV-Cache in Distributed LLM Serving." In: *Proceedings of the 38th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. doi:10.1145/3816782.3819214.

Z. Zhang, H. Li, Y. Zhao, C. Lin, and J. Liu. 2023. "POS: An Operator Scheduling Framework for Multi-Model Inference on Edge Intelligent Computing." In: *Proceedings of the ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*.

J. Zhao et al. 2022. "Apollo: Automatic Partition-based Operator Fusion through Layer by Layer Optimization." In: *Proceedings of Machine Learning and Systems (MLSys)*.

J. Zhou, Y. Chen, Z. Hong, W. Chen, Y. Yu, T. Zhang, H. Wang, C. Zhang, and Z. Zheng. 2024. "Training and Serving System of Foundation Models: A Comprehensive Survey." *IEEE Open Journal of the Computer Society*, 5. doi:10.1109/OJCS.2024.3380828.

X. Zhu et al. 2026. "QiMeng-Kernel: Macro-Thinking Micro-Coding Paradigm for LLM-Based High-Performance GPU Kernel Generation." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 40. doi:10.1609/aaai.v40i34.40155.

A Survey Corpus of 93 Papers

Table 7 summarizes how the surveyed papers were assigned to primary topic categories for Figure 1; the enumerated list below is the complete corpus, sorted by publication year. Each entry corresponds to a work cited in the main text.

Table 7. Primary-topic assignment of the 93 surveyed papers (matches Figure 1).

Primary topic	Count
Model & quantization	25
Serving systems	22
Edge & heterogeneous hardware	25
Evaluation & MLOps	16
Security & compliance	5
Total	**93**

(1) [Wohlin 2014]
(2) [Goodfellow et al. 2015]
(3) [Abadi et al. 2016]
(4) [Kasture and Sanchez 2016]
(5) [Tramèr, F. Zhang, et al. 2016]
(6) [Jouppi, Young, et al. 2017]
(7) [Shokri et al. 2017]
(8) [Umuroglu et al. 2017]
(9) [Boehm et al. 2018]
(10) [Berger et al. 2018]
(11) [Chen et al. 2018]
(12) [Chard et al. 2019]
(13) [Tramèr and Boneh 2019]
(14) [Bhardwaj et al. 2020]
(15) [Reddi, Cheng, et al. 2020]
(16) [Jahanshahi et al. 2020]
(17) [Ghadani et al. 2020]
(18) [Haroush et al. 2020]
(19) [Marco et al. 2020]
(20) [H. Wu et al. 2020]

(21) [Lechner and Jantsch 2021]
(22) [Seo et al. 2021]
(23) [Wang et al. 2021]
(24) [M. Li et al. 2021]
(25) [Shafi et al. 2021]
(26) [Dacrema et al. 2021]
(27) [Hubara et al. 2021]
(28) [Lattner et al. 2021]
(29) [Liu et al. 2021]
(30) [Niu et al. 2021]
(31) [Brewer et al. 2021]
(32) [Romero et al. 2021]
(33) [Copik et al. 2021]
(34) [Almeida et al. 2021]
(35) [Tailor et al. 2021]
(36) [Verma et al. 2021]
(37) [Z. Yang et al. 2021]
(38) [Manzoor et al. 2022]
(39) [L. Xie et al. 2022]
(40) [D. Xiao et al. 2022]
(41) [Jeong et al. 2022]
(42) [Yu et al. 2022]
(43) [Gholami et al. 2022]
(44) [Y. Yang et al. 2022]
(45) [Reddi, Kanter, et al. 2022]
(46) [Y. Wu et al. 2022]
(47) [Yao et al. 2022]
(48) [Wei et al. 2022]
(49) [Emani et al. 2022]
(50) [Z. Xu et al. 2022]
(51) [Zhao et al. 2022]
(52) [Koubaa et al. 2022]
(53) [Tornede et al. 2023]
(54) [Firoozshahian et al. 2023]
(55) [Jouppi, Kurian, et al. 2023]
(56) [Olszewski et al. 2023]
(57) [Z. Zhang et al. 2023]
(58) [Pappula 2023]
(59) [Salmani et al. 2023]
(60) [Z. Li et al. 2023]
(61) [Spector and Ré 2023]
(62) [F. Xu et al. 2023]
(63) [B. Li et al. 2024]
(64) [Shankar 2024]
(65) [Bondarenko et al. 2024]
(66) [Jajal et al. 2024]

(67) [H. Xia et al. 2024]
(68) [Svirschevski et al. 2024]
(69) [Shinde 2024]
(70) [Gagrani et al. 2024]
(71) [Shi et al. 2024]
(72) [Fu et al. 2024]
(73) [Moon et al. 2024]
(74) [W. Xiao et al. 2024]
(75) [C. Xia et al. 2024]
(76) [Ahmad et al. 2024]
(77) [A. Agrawal et al. 2024]
(78) [Sun et al. 2024]
(79) [B. Wu et al. 2024]
(80) [Sheng et al. 2024]
(81) [Razavi et al. 2024]
(82) [Gao et al. 2024]
(83) [Lindkvist 2024]
(84) [Zhou et al. 2024]
(85) [Rachmanto et al. 2024]
(86) [Sanovar et al. 2025]
(87) [Musa et al. 2025]
(88) [H. Li et al. 2025]
(89) [Samarasinghe 2025]
(90) [Sander et al. 2025]
(91) [Zhu et al. 2026]
(92) [Nellutla 2026]
(93) [Yüzügüler et al. 2026]

Received 02 March 2026; accepted 16 October 2026