World Journal of Information Technology
Print ISSN: 2959-9903
Online ISSN: 2959-9911
DOI: https://doi.org/10.61784/wjit4125

XSLA: AN SLA-AWARE HETEROGENEOUS XPU SCHEDULING FRAMEWORK FOR EDGE AI INFERENCE

TingJie Chen[1], WenBin Shang[2*], ZiXuan Guo[3]
[1]*Intel, Shanghai 200333, China.*
[2]*University of Glasgow, Glasgow G12 8QQ, U.K.*
[3]*New York University, NY 10012, United States.*
[*]*Corresponding Author: WenBin Shang*

Abstract: Edge AI deployments increasingly aggregate several classes of accelerator — embedded GPUs, neural processing units, FPGAs, dataflow ASICs and ARM CPUs — into a single "XPU" pool, while the requests they serve arrive with heterogeneous service-level agreements (SLAs). Using published measurements of 15,625 architectures on six real accelerator platforms, we show that one fixed model varies in latency by up to 51.9x across XPU classes, that 30.1% of architectures cannot execute on the Edge TPU at all, and that no accelerator is simultaneously best for latency, energy and accuracy. Which accelerator to use and which model variant to run are therefore one decision, not two. We present XSLA, which co-selects variant and accelerator per request. XSLA (i) shrinks the candidate set with a per-accelerator Pareto test we prove is lossless, (ii) chooses among deadline-feasible candidates by a drift-plus-penalty rule whose quality weight adapts to measured backlog, and (iii) estimates backlog per SLA class so latency-critical traffic is never queued behind best-effort work. In a measurement-driven simulator with 6 XPU classes, 3 tasks and 24 real variants, XSLA holds SLA violation at 0.0% up to 9,000 req/s, where an accuracy-greedy model-less policy misses 47.0% of deadlines and fixed-model EDF misses 99.0%. It raises deadline-constrained accuracy yield by 14.4–81.4% over the strongest baseline while using 37.4–58.4% less energy per request, lifting pool utilisation from 0.26 to 0.99.
Keywords: Edge computing; Inference serving; Heterogeneous accelerators; Service-level agreements; Real-time scheduling; Energy efficiency

1 INTRODUCTION

Inference is moving to the edge because bandwidth, privacy and tail latency all argue against a round trip to a datacentre [1-2]. What has changed recently is not only where inference runs but what it runs on. A camera gateway, a factory cell or a vehicle head-unit now routinely contains more than one kind of accelerator: an embedded GPU, a USB or on-package neural processing unit (NPU), sometimes an FPGA or a dataflow ASIC, and always one or more general-purpose CPU clusters. The industry shorthand for this pool of dissimilar compute is XPU.

Serving requests on an XPU pool is harder than serving them on a rack of identical GPUs, for two reasons that compound. First, performance heterogeneity is extreme and not monotone: an accelerator that is fastest for one network can be mediocre for another, and the accelerator that minimises latency is frequently not the one that minimises energy. Second, an operator does not deploy one model; it deploys a family of variants that trade accuracy for cost. Existing serving systems address at most one half of this. Model-less serving systems such as INFaaS pick a variant but assume interchangeable hardware [3]. Heterogeneity-aware cluster schedulers such as Gavel pick hardware but treat the model as fixed [4]. On an edge XPU pool, fixing either half throws away most of the operating envelope.

This paper argues that variant selection and accelerator selection are a single decision under an SLA constraint, and it makes that decision cheap enough to take per request. We ground the argument in real measurements rather than analytical cost models: we build a serving testbed from HW-NAS-Bench, which reports measured latency and energy for 15,625 NAS-Bench-201 architectures on six accelerator platforms, and pair it with the corresponding measured accuracies [5-6]. Three facts emerge that any edge scheduler must confront (Section 3): one model's latency spans 51.9x across XPU classes; 30.1% of architectures cannot run on the Edge TPU because of operator-coverage gaps; and only 0.4–0.7% of deployable architectures lie on the accuracy–latency–energy Pareto frontier.

We then present XSLA, an SLA-aware heterogeneous XPU scheduling framework. Its contributions are:

1) A joint variant–accelerator co-selection formulation for edge inference with per-class latency SLAs, in which the served accuracy is an outcome of scheduling rather than an input to it.

2) A per-accelerator Pareto pruning test we prove is lossless (Lemma 1): it never removes a candidate that could have been chosen, for any backlog or deadline. It removes 24.0% of candidate pairs for an 8-variant zoo and 57.8% for a 32-variant zoo, cutting enumeration cost 1.27x–2.04x.

3) A drift-plus-penalty dispatch rule with a load-adaptive quality weight. Adaptation is what keeps the scheduler stable: a static weight yields 28.0% deadline violations at 9,000 req/s where adaptation yields 0.0%, and it cuts hyper-parameter sensitivity by 1.6x.

4) Class-aware backlog estimation, making the feasibility test consistent with the priority discipline the device queues actually use. Under bursty arrivals it holds latency-critical violations at 0.0–3.2% while every baseline exceeds 24%.

5) Deadline-constrained accuracy yield (DCAY), a single metric that credits accuracy only for requests that meet their deadline, so that accuracy and timeliness cannot be traded silently against each other.
An extensive evaluation (Section 6) covers offered-load sweeps, bursty and diurnal arrivals, component ablations, hyper-parameter robustness, unmodelled device slowdown, six pool compositions, SLA tightness, dispatch cost, and sensitivity to our two modelling assumptions. We report negative results as well: online latency re-calibration, which we expected to matter, contributes almost nothing, because queue feedback already absorbs systematic slowdown.

2 RELATED WORK

2.1 SLO-Aware Inference Serving

Clipper introduced latency-aware batching and model selection for prediction serving [7]; Clockwork achieves predictable tail latency by making the whole stack deterministic [8]; Nexus batches across models on GPU clusters [9]; MArk exploits elastic cloud resources to meet SLOs cost-effectively [10]; SHEPHERD uses request-level multiplexing to raise goodput under bursty demand [11]. INFaaS is closest to our variant dimension: it selects a variant to satisfy an accuracy/latency query without the user naming a model [3]. All target datacentre GPUs, and their central mechanism is batching, which is largely unavailable on the fixed-function edge NPUs and ASICs we study; none treats accelerator class and variant as a coupled choice.

2.2 Edge Inference and Adaptation

Neurosurgeon partitions a network between device and cloud [2]; split-computing and early-exit approaches generalise this [12]. DeepDecision and Jellyfish adapt input resolution and network depth to network conditions to hold an end-to-end SLO, and Jellyfish in particular couples DNN adaptation with user mapping [13-14]. On-device multi-DNN execution has received sustained attention: RT-mDL schedules mixed real-time DL tasks, Band coordinates multiple DNNs across heterogeneous mobile processors, CoDL co-executes a single network on CPU and GPU, DREAM schedules dynamic real-time multi-model workloads, and Sparse-DySta combines static and dynamic scheduling for sparse multi-DNN workloads [15-19]. These works schedule across heterogeneous processors within one device and mostly keep the model fixed; we schedule across a pool of independent accelerators and treat the deployed variant as a decision variable with an explicit energy objective.

2.3 Heterogeneity-Aware Cluster Scheduling

Gavel expresses cluster policies as optimisation problems accounting for per-accelerator throughput heterogeneity, showing that heterogeneity-awareness lets a cluster sustain higher load [4]; Weng et al. characterise a production heterogeneous GPU cluster [20]. Both target training jobs with second-to-hour lifetimes, whereas our requests complete in 0.3–70 ms, so the per-decision budget is microseconds. From classical scheduling we inherit EDF as the queue discipline within a class and the earliest-finish-time heuristic of HEFT as a baseline dispatch rule [21-22]; our optimisation follows the drift-plus-penalty method [23].

2.4 Hardware-Aware Benchmarks and Model Families

Our measurement substrate is HW-NAS-Bench, which reports measured latency (and, on three platforms, measured energy) for every NAS-Bench-201 architecture on an Edge GPU, Edge TPU, Raspberry Pi 4, Pixel 3, the Eyeriss ASIC and an FPGA [5-6, 24]. nn-Meter predicts latency on diverse edge devices [25]; MLPerf Inference standardises measurement methodology [26]. On the model side, families explicitly parameterised for accuracy–cost trade-offs — MobileNetV2, EfficientNet, once-for-all networks — are what make a variant zoo practical to deploy [27-29]. We use these bodies of work as inputs; our contribution is the scheduler that exploits them online.

3 MEASUREMENT-DRIVEN CHARACTERISATION

3.1 Data Sources and What Is Measured

We combine two public artefacts. HW-NAS-Bench v1.0 gives, for each of the 15,625 architectures in the NAS-Bench-201 search space and each of three datasets, the measured single-image inference latency on six hardware platforms, plus measured energy on three of them [5]. The compact NAS-Bench-201 release of [6] gives the corresponding classification accuracies. Both index architectures identically; as a consistency check, the Spearman correlation between accuracy and measured Edge GPU latency is 0.26–0.30 across the three datasets, i.e. positive as expected for an accuracy–cost frontier.
Two modelling assumptions must be stated plainly. First, the accuracies we use come from the 12-epoch training budget of NAS-Bench-201, so absolute values are lower than the 200-epoch figures usually quoted (89.16% rather than 94.37% for the best CIFAR-10 cell); what our scheduler consumes is the relative ordering and spacing of variants, which the shorter budget preserves. Second, HW-NAS-Bench reports energy only for the Edge GPU, Eyeriss and FPGA. For the Edge TPU, Raspberry Pi 4 and Pixel 3 we estimate energy as $E = P \cdot t$ using published device-level active power (2.0 W,

4.0 W and 2.5 W respectively). We mark these as estimates in Table 1 and vary them by ±30% to confirm that no conclusion depends on them.

Table 1 The Heterogeneous XPU Pool

XPU class	Hardware platform	In pool	Mean lat. (ms)	Lat. range (ms)	Mean energy (mJ)	Energy source
GPU (EdgeGPU)	NVIDIA Jetson TX2 GPU	2	3.85	1.44 – 6.98	17.20	measured [5]
NPU (EdgeTPU)	Google Edge TPU (Coral)	3	0.63	0.35 – 1.36	1.26	P·t, 2.0 W
ASIC (Eyeriss)	Eyeriss dataflow accelerator [24]	1	2.81	0.42 – 7.83	0.56	measured [5]
FPGA	FPGA DNN accelerator	2	2.53	0.28 – 6.68	17.70	measured [5]
CPU (RasPi4)	Raspberry Pi 4B, ARM Cortex-A72	3	14.35	1.02 – 70.26	57.41	P·t, 4.0 W
Mobile CPU	Pixel 3, Snapdragon 845 (TFLite)	1	6.26	0.42 – 34.69	15.65	P·t, 2.5 W

Note: Latency and energy are means over the 24 deployed variants; energy source distinguishes HW-NAS-Bench measurements from P·t estimates.

Only the arrival process is synthetic: Poisson for the main sweep, plus a two-state Markov-modulated Poisson process and a non-stationary diurnal process for robustness, since production serving traces exhibit both burstiness and strong diurnal structure [20, 30].

3.2 Four Observations That Shape the Design

O1: heterogeneity is extreme and non-monotone. Averaged over the 24 variants we deploy (Table 1), mean per-inference latency ranges from 0.63 ms on the Edge TPU to 14.35 ms on the Raspberry Pi 4 — a 22.8x span — and for individual variants the fastest-to-slowest ratio across XPU classes reaches 51.9x. Crucially, the ranking is metric-dependent: the Edge TPU is the fastest platform but Eyeriss is 2.2x more energy-efficient per inference (0.56 mJ vs 1.26 mJ). A scheduler that optimises only latency therefore leaves large energy savings on the table, and vice versa.
O2: not every model runs on every accelerator. 4,699 of the 15,625 architectures (30.1%) have no valid Edge TPU measurement, reflecting operator-coverage gaps in the compiler toolchain. Deployability is therefore a hard constraint, not a soft cost, and a scheduler must carry a per-(variant, XPU) compatibility mask.
O3: the useful frontier is tiny. Of the architectures deployable on at least one XPU, only 72 of 15,284 (CIFAR-10), 54 of 15,045 (CIFAR-100) and 76 of 10,899 (ImageNet16-120) are Pareto-optimal in (accuracy, best-case latency, best-case energy) — between 0.4% and 0.7%. A production zoo is consequently small, which is what makes per-request enumeration over variants affordable at all.
O4: variant choice is a capacity decision. Within one task the most accurate variant costs 3.7x more Edge GPU time and 19.9x more Raspberry Pi time than the least accurate one. Serving a heavy variant therefore does not merely spend energy, it removes throughput from the pool. This is why accuracy cannot be maximised greedily per request: doing so silently reduces the load the pool can carry, and the deadline misses appear later and elsewhere, see Figure 1 and 2.

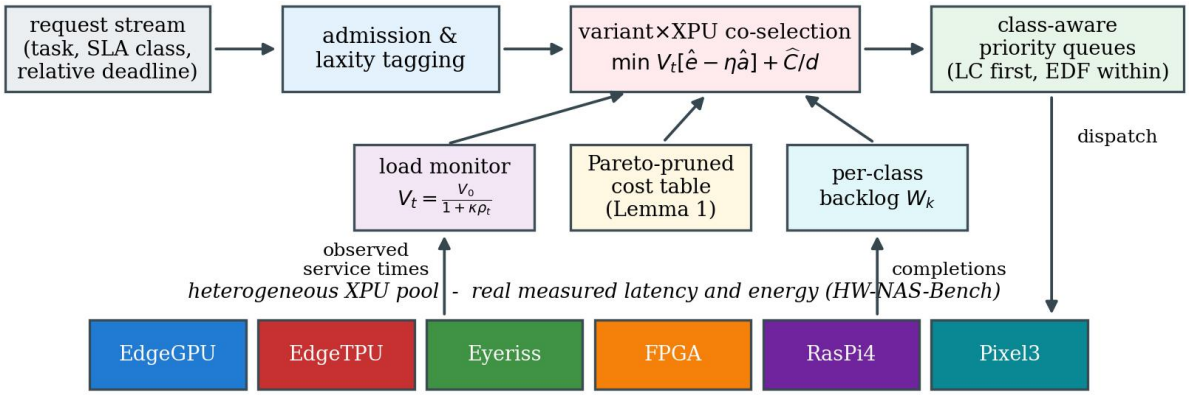

Figure 1 XSLA

Note: The co-selection engine scores deadline-feasible (variant, XPU) pairs from a statically Pareto-pruned table using per-class backlog and a load-adaptive quality weight; the selected pair is enqueued under class priority with EDF inside each class. Completions feed the backlog estimator and the load monitor.

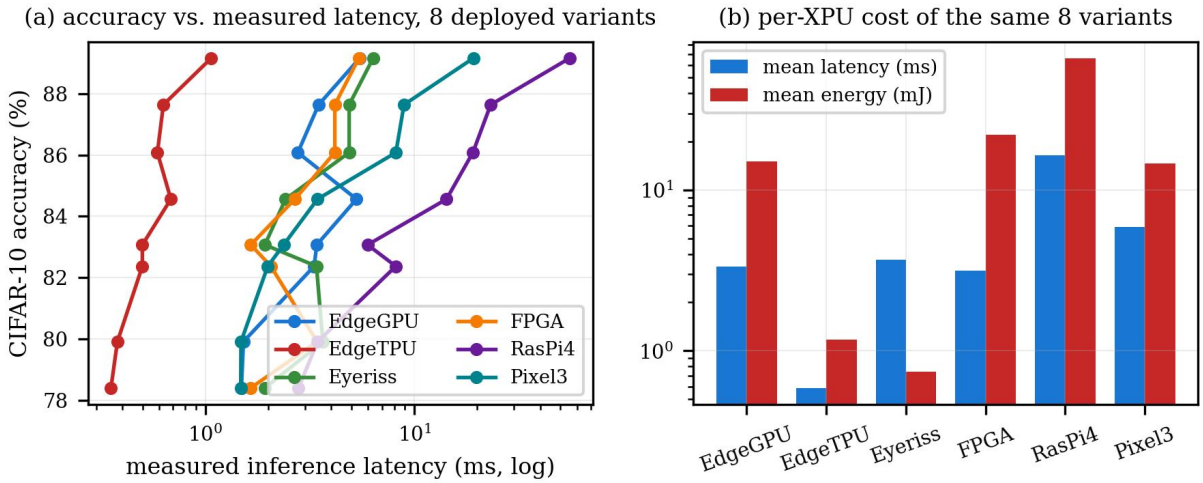

Figure 2 The Deployed Zoo, from Real Measurements

Note: (a) Accuracy against measured latency for the eight CIFAR-10 variants on each XPU class: the frontier is a different shape on every accelerator, and the Raspberry Pi curve is two orders of magnitude to the right of the Edge TPU curve. (b) Mean latency and mean energy of the same 24 variants per XPU class; note that the latency-optimal accelerator (Edge TPU) is not the energy-optimal one (Eyeriss).

4 SYSTEM MODEL AND PROBLEM STATEMENT

An edge site owns K accelerators. Accelerator k belongs to XPU class c(k) from Table 1 and 2, which fixes its performance. Each executes one inference at a time — the fixed-function NPUs, ASICs and FPGAs we model do not time-share — and maintains a local queue.

Table 2 Deployed Variant Zoo for Task T1 (CIFAR-10)

Variant	NB-201 idx	Acc. (%)	EdgeGPU	EdgeTPU	Eyeriss	FPGA	RasPi4	Pixel3
T1-v0	4512	78.40	1.486	0.353	1.925	1.648	2.789	1.476
T1-v1	14550	79.91	1.520	0.379	3.646	3.421	3.443	1.480
T1-v2	6435	82.37	3.312	0.499	3.400	2.058	8.152	1.987
T1-v3	12740	83.07	3.412	0.499	1.925	1.648	5.975	2.368
T1-v4	6064	84.56	5.285	0.683	2.417	2.679	14.256	3.439
T1-v5	3603	86.08	2.769	0.589	4.874	4.163	19.058	8.160
T1-v6	15029	87.64	3.481	0.629	4.874	4.163	23.187	8.922
T1-v7	5374	89.16	5.477	1.066	6.349	5.421	55.376	19.166

Note: All eight variants are Pareto-optimal in (accuracy, latency, energy); accuracies are NAS-Bench-201 12-epoch test accuracies, latencies are HW-NAS-Bench measurements in ms. Tasks T2 (CIFAR-100) and T3 (ImageNet16-120) are constructed identically.

The site serves tasks T. For task τ the operator has deployed a zoo $V(\tau)$ of model variants. Variant v has accuracy a(v), and on accelerator k it has measured latency $\ell(v, k)$ and energy e(v, k), both defined only if v is deployable on class c(k); the compatibility mask is $M(v, k) \in \{0, 1\}$.

Request r has arrival time tr, task τr and SLA class $\chi r \in \{LC, BE\}$, with class-specific relative deadline $D\chi$ giving dr = tr + $D\chi r$; we use DLC = 10 ms, DBE = 50 ms, and 40% latency-critical load. At tr, without knowledge of future arrivals, the scheduler must choose a pair (vr, kr) with M(vr, kr) = 1 and a queue discipline. Let fr = 1 if r completes by dr.

Three quantities matter simultaneously: timeliness, served accuracy and energy. Reporting them separately invites a scheduler to look good on one by quietly sacrificing another, so we fold the first two into a single primary metric.

Definition 1 (Deadline-constrained accuracy yield). For a request set R, DCAY = $(1/|R|) \Sigma a(vr) \cdot fr$.

DCAY is the mean accuracy delivered on time: a late request contributes nothing however accurate, and a fast request contributes only its variant's accuracy. It is bounded above by the best deployed variant, so the gap to that bound measures everything lost to contention and degradation together. Our objective is

maximise DCAY subject to bounded queues, and, among schedules of equal DCAY, minimise mean energy per request.

The decision space per request is $|V(\tau)| \times K$ — 96 pairs in our default configuration — and the decision must be taken in microseconds. Section 5 builds a rule that respects that budget.

5 THE XSLA FRAMEWORK

5.1 Overview

Figure 1 shows the structure. An arriving request is tagged with its class and deadline; the co-selection engine scores every surviving (variant, accelerator) candidate from three pieces of state — a static Pareto-pruned cost table, per-class

backlog, and a quality weight driven by a load monitor — and the chosen pair is enqueued under class priority. Completions feed the backlog estimator and the load monitor.

5.2 Lossless Per-Accelerator Pareto Pruning

Enumerating all |V| × K pairs per request is wasteful, but pruning must not change the outcome. The following test is safe because it compares variants only within one accelerator, where the backlog term is common to all of them.

Lemma 1. Fix accelerator k and variants u, v with M(u,k) = M(v,k) = 1. If a(v) ≥ a(u), ℓ(v,k) ≤ ℓ(u,k) and e(v,k) ≤ e(u,k), then for every backlog $W_k \geq 0$, every remaining deadline d > 0 and all weights V_t, η ≥ 0, candidate (u,k) is never uniquely optimal for the rule of Section 5.4. Removing (u,k) cannot reduce the attained objective.

Proof. The predicted completion of (v,k) is $\hat{C}(v,k) = W_k + \ell(v,k) \leq W_k + \ell(u,k) = \hat{C}(u,k)$. Hence whenever (u,k) satisfies the feasibility test $\hat{C} \leq d$, so does (v,k). The score is $J = V_t[\hat{e} - \eta\,\hat{a}] + \hat{C}/d$ with ê, â the normalised energy and accuracy. Since ê(v,k) ≤ ê(u,k), â(v) ≥ â(u) and V_t, η ≥ 0, each of the three terms of J(v,k) is no larger than the corresponding term of J(u,k), so J(v,k) ≤ J(u,k). A minimiser therefore always exists in the pruned set.

The restriction to a fixed accelerator is essential: a cross-accelerator Pareto filter would be unsound, because a statically dominated pair on an idle accelerator can beat a statically dominating pair on a congested one. Pruning is computed once at deployment, so it costs nothing online. Section 6.7 measures its effect, and the ablation in Table 4 confirms empirically that pruned and unpruned XSLA produce bit-identical outcomes, as Lemma 1 predicts.

5.3 Class-Aware Backlog Estimation

The feasibility test is only as good as its backlog estimate, and the estimate must match the discipline the queues actually use. XSLA runs strict class priority (LC before BE) with EDF inside each class [21]. Accordingly it tracks two backlogs per accelerator: Q_k^{LC} and Q_k^{BE}, the queued service time of each class. An arriving LC request will overtake all queued BE work, so its predicted wait is $W_k = \rho_k + Q_k^{LC}$, where ρ_k is the residual service of the running inference; a BE request must additionally wait for Q_k^{BE}. Using one class-agnostic backlog for both, as a FIFO scheduler must, makes the LC estimate pessimistic and the BE estimate optimistic at the same time; Table 4 shows this costs 1.8% of DCAY at 9,000 req/s.

5.4 Drift-Plus-Penalty Co-Selection

Let W(k) be the class-aware backlog above, $\hat{C}(v,k) = W(k) + \ell(v,k)$ the predicted completion delay, and d = dr − t the remaining laxity. XSLA first forms the feasible set $F = \{(v,k) : M(v,k) = 1, \text{pruned}, \hat{C}(v,k) \leq d\}$ and, if F is non-empty, selects

$$(v_r, k_r) = \arg\min_{(v,k)\in F} V_t \cdot [\hat{e}(v,k) - \eta \cdot \hat{a}(v)] + \frac{\hat{C}(v,k)}{d} \quad (1)$$

The second term is the fraction of the request's laxity that the choice would consume: it is the drift, and it is what steers work away from congested accelerators and away from variants that are slow on the accelerator under consideration. The first term is the penalty: normalised energy minus normalised accuracy, weighted by Vt. Two design points deserve comment.

First, accuracy is normalised on an absolute scale, â(v) = (a(v) − amin(τ))/10, in units of ten accuracy points, not by the zoo's own min–max range. Per-zoo normalisation would make a zoo whose variants differ by 2 points behave like one whose variants differ by 17; we observed exactly that failure, and the absolute scale removes it (Section 6.12).

Second, the division by d makes the rule class-adaptive for free. For an LC request d is 10 ms, so a 4 ms predicted delay consumes 40% of the budget and dominates the penalty term; for a BE request the same delay consumes 8% and quality wins. One rule therefore yields conservative behaviour for tight deadlines and quality-seeking behaviour for loose ones, without separate policies per class.

5.5 Load-Adaptive Quality Weight

Vt controls how much accuracy the scheduler is willing to buy with device time. A fixed value cannot work across a load range: the value that maximises accuracy at 3,000 req/s over-commits the pool at 9,000 req/s. XSLA therefore drives Vt from a smoothed measure of pool congestion,

$$\rho_t = (1-\beta)\cdot\rho_{t-1} + \beta \cdot \min\left(1, \frac{\bar{W}_t}{D_{BE}}\right), \quad V_t = \frac{V_0}{1+\kappa\cdot\rho_t} \quad (2)$$

with $\bar{W}t$ the mean backlog across accelerators, β = 0.02 and κ = 20. As the pool fills, Vt shrinks and the drift term takes over, so the scheduler slides continuously from "most accurate variant that fits" to "cheapest variant that fits". This is the drift-plus-penalty structure of [23], with V adapted rather than fixed; the price of adaptation is that the classical O(1/V) optimality gap is no longer constant, which is why we characterise it empirically (Section 6.6) instead of claiming a bound.

5.6 Overload Handling

If F is empty, no deployable pair can meet the deadline. XSLA then minimises lateness by choosing the pair with the smallest $\hat{C}$, except that a BE request whose best predicted completion exceeds $\gamma \cdot d$ ($\gamma = 2$) is shed, freeing capacity for traffic that can still be served. LC requests are never shed. This is the mechanism behind the class isolation observed under bursty arrivals in Section 6.8.

5.7 Online Re-Calibration and Cost

The estimate $\ell(v,k) = s_k \cdot \ell(v,k)$ carries a per-accelerator multiplier updated by an exponentially weighted average of observed-to-nominal service time ($\alpha = 0.05$), intended to absorb thermal throttling or co-runner interference. Algorithm 1 summarises one dispatch; its cost is $O(|VP(\tau)| \cdot K)$ plus $O(\log n)$ for the queue insertion, measured at 29 µs in unoptimised vectorised Python at $K = 12$ and 37 µs at $K = 96$ (Section 6.7).

Algorithm 1 XSLA Dispatch of One Request r

```
input: task τ, class χ, deadline d_r, arrival t
W ← ρ + Q^LC + (Q^BE if χ = BE else 0)            ▷ per accelerator
Ĉ ← W ⊕ ℓ(·,·)                                    ▷ pruned pairs only
F ← {(v,k) : M(v,k)=1, pruned, Ĉ(v,k) ≤ d_r − t}
if F ≠ ∅ then
      (v,k) ← arg min_F  V_t[ê − η â] + Ĉ/(d_r − t)
else
      (v,k) ← arg min over pruned pairs of Ĉ
      if χ = BE and Ĉ(v,k) > γ(d_r − t) then shed r; return
enqueue r at k with key (χ, d_r);   Q^χ_k += ℓ(v,k)
ρ_t ← (1−β)ρ_t + β·min(1, W̄/D_BE);   V_t ← V_0/(1+κρ_t)
on completion: s_k ← (1−α)s_k + α·(observed/ℓ(v,k))
```

6 EVALUATION

6.1 Setup

We implement a discrete-event simulator in which every service time is drawn from the HW-NAS-Bench measurement for the selected (variant, accelerator) pair, perturbed by a log-normal factor with $\sigma = 0.12$ to represent run-to-run variation. The pool is the 12 accelerators of Table 1. The zoo holds 8 variants per task for 3 tasks (24 variants); the task mix is 50/30/20 across CIFAR-10, CIFAR-100 and ImageNet16-120. 40% of requests are latency-critical with a 10 ms deadline, the rest best-effort with 50 ms. Each configuration runs 15,000 requests and is averaged over 3 seeds. XSLA uses $V0 = 0.1$, $\eta = 1.0$, $\kappa = 20$, $\gamma = 2$, $\alpha = 0.05$, selected once by mean DCAY over 3/6/9k req/s and held fixed for every experiment reported here.

We compare against six policies spanning the design space: Round-Robin and JSQ (fixed most accurate variant, load-oblivious and load-aware placement); EDF+EFT (fixed variant, earliest-finish-time dispatch with EDF queues, i.e. the classical real-time baseline in the spirit of [21-22]); Latency-Greedy (minimum predicted completion over all pairs); Energy-Greedy (minimum energy among deadline-feasible pairs); and Accuracy-Greedy (maximum accuracy among deadline-feasible pairs, the model-less selection strategy of [3] adapted to our setting). The last three, like XSLA, may switch variants; the first three may not.

6.2 Offered-Load Sweep

Figure 3 and Table 3 give the main result. Up to 3,000 req/s the pool is under-subscribed and the fixed-variant schedulers are optimal: EDF+EFT and Accuracy-Greedy both deliver DCAY 70.52, the ceiling set by the best deployed variants. XSLA delivers 69.37, 1.6% below the ceiling, but at 1.86 mJ per request against their 3.57 mJ — a 48.0% energy saving for a 1.6% accuracy concession, which is the trade the operator asked for by setting V0.

Beyond 3,000 req/s the picture inverts. EDF+EFT and JSQ collapse: at 4,500 req/s they miss 96.0% and 97.5% of deadlines (DCAY 2.80 and 1.77), because a fixed heavy variant cannot be served at that rate by this pool. Accuracy-Greedy degrades more gracefully, since it can fall back to lighter variants, but still reaches 24.2% violations at 6,000 req/s and 47.0% at 9,000: choosing the most accurate feasible variant per request in isolation commits more device time than the pool has, and the resulting backlog invalidates the feasibility test for everything behind it — observation O4 in action.

XSLA holds violations at exactly 0.0% for both SLA classes across the entire range to 9,000 req/s, and 0.1% at 10,500 req/s, while its goodput tracks offered load one-for-one (8,984 req/s served on time at 9,000 offered). Its DCAY declines gently and monotonically — 69.4, 68.7, 67.7, 66.6, 65.2, 63.5 — which is precisely the graceful degradation the design targets: as load rises the scheduler spends accuracy, not deadlines.

Latency-Greedy is the only baseline that never violates an SLA, and it is the right comparison for the value XSLA adds. It always runs the smallest variant on the fastest accelerator, so its DCAY is pinned at 56.98 regardless of load and its pool utilisation never exceeds 0.33 (Table 3). XSLA converts that idle capacity into accuracy: at 9,000 req/s it runs the

pool at 0.99 utilisation and delivers 65.16 DCAY, 14.4% more, with identical timeliness; against Accuracy-Greedy the margin is 81.4%. Energy tells the same story: at 6,000 req/s XSLA uses 4.56 mJ per request against 7.28 mJ for Accuracy-Greedy (−37.4%), 10.95 mJ for EDF+EFT (−58.4%) and 12.28 mJ for JSQ (−62.9%). Summarising by SLA-compliant capacity — the highest load at which violations stay under 1% — XSLA exceeds 10,500 req/s while Accuracy-Greedy, EDF+EFT and JSQ all stop at 3,000 req/s, a factor of at least 3.5.

Table 3 Main Comparison under Poisson Arrivals, 15,000 Requests × 3 Seeds

Policy	DCAY 3k	DCAY 6k	DCAY 9k	Viol. 3k	Viol. 6k	Viol. 9k	mJ/req 3k	mJ/req 6k	mJ/req 9k	Util. 9k
Round-Robin	17.85	17.72	17.70	74.6%	74.8%	74.8%	67.41	67.41	67.41	0.34
JSQ	68.87	0.96	0.67	2.2%	98.6%	99.0%	13.77	12.28	12.26	0.99
EDF+EFT	70.52	1.17	0.71	0.0%	98.3%	99.0%	3.57	10.95	10.97	1.00
Energy-Greedy	55.94	56.09	55.39	1.6%	1.6%	2.7%	0.59	0.62	0.63	0.31
Latency-Greedy	56.98	56.98	56.98	0.0%	0.0%	0.0%	0.97	0.96	1.01	0.26
Accuracy-Greedy	70.52	51.57	35.92	0.0%	24.2%	47.0%	3.57	7.28	4.92	1.00
XSLA (ours)	69.37	67.75	65.16	0.0%	0.0%	0.0%	1.86	4.56	4.82	0.99

Note: DCAY is the primary metric (higher is better); violation is over both SLA classes. Best value per column in the shaded row.

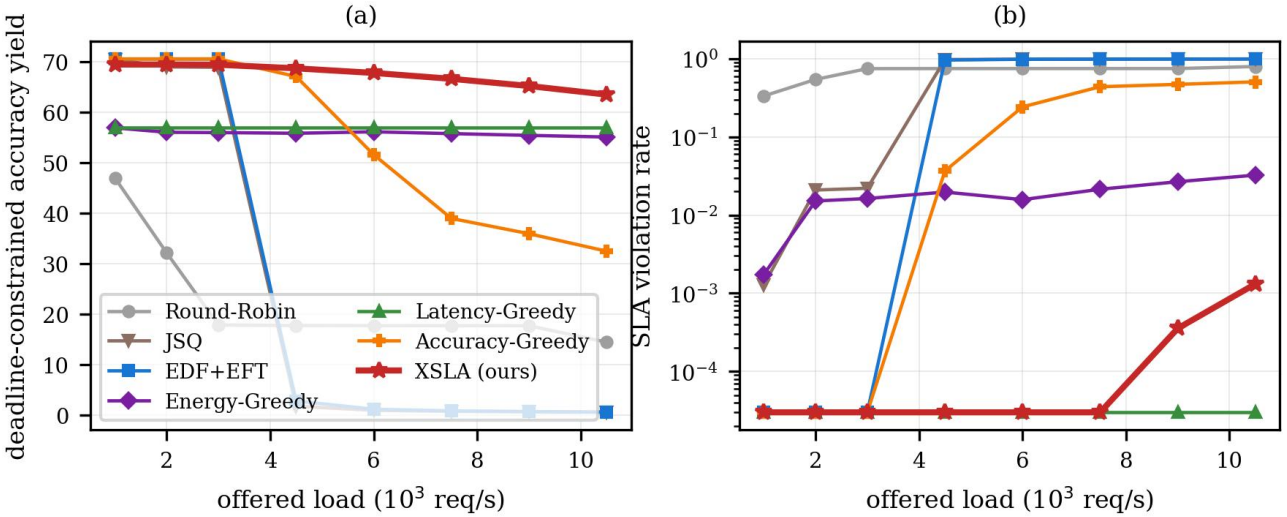

Figure 3 Offered-Load Sweep

Note: (a) Deadline-constrained accuracy yield. (b) SLA violation rate (log scale; XSLA is clamped at the plot floor because it is exactly zero up to 9,000 req/s). The fixed-variant schedulers (EDF+EFT, JSQ) fall off a cliff between 3,000 and 4,500 req/s; Accuracy-Greedy degrades but violates deadlines; Latency-Greedy is timely but capped in accuracy. XSLA is the only policy that is simultaneously timely and accurate across the range. Energy per request and pool utilisation for the same sweep are given in Table 3.

6.3 Where the Extra Accuracy Comes From

Figure 4(a) shows which XPU class XSLA selects as load rises. At 1,000 req/s, 99.8% of requests go to the Edge TPU, the fastest and second most energy-efficient class. As the Edge TPUs saturate the scheduler spills in a specific order: first to the FPGA and Eyeriss (energy-efficient, moderate latency), then to the Edge GPU and Pixel 3, and only above 7,500 req/s to the Raspberry Pi 4, which is 22.8x slower and 45.6x less energy-efficient. By 9,000 req/s the mix is 48.4% NPU, 12.8% FPGA, 14.3% CPU, 9.7% GPU, 7.6% ASIC and 7.2% mobile CPU. No hand-written priority list produces this ordering: it falls out of the score, and it explains the utilisation gap in the last column of Table 3.

6.4 The Accuracy–Energy Knob

Figure 4(b) sweeps η, the accuracy weight, at three loads. The trace is a clean Pareto frontier: at 6,000 req/s, η = 0 gives 56.98% accuracy at 0.78 mJ, η = 0.5 gives 67.18% at 2.44 mJ, and η = 3 gives 68.30% at 6.27 mJ. Diminishing returns set in sharply after η = 1: the last 0.5 accuracy points cost 37% more energy. Above η = 1.5 the frontier also becomes unsafe at high load — the circled points in Figure 4(b) violate more than 1% of deadlines at 9,000 req/s — which gives the operator a clear rule: increase η until either energy or the SLA margin becomes unacceptable.

6.5 Component Ablation

Table 4 removes one mechanism at a time at 9,000 req/s. Disabling variant switching is catastrophic — DCAY falls from 65.16 to 26.45 and violations rise to 62.3% — confirming that variant–accelerator co-selection, not accelerator selection alone, is what carries the result. Freezing the quality weight is the second most damaging change: 28.0% violations and DCAY 47.53, because a static weight keeps buying accuracy after the pool is full. Replacing class-aware

queues and backlog with plain FIFO costs 1.8% of DCAY; the effect is modest in aggregate but, as Section 6.8 shows, it is what protects latency-critical traffic under bursts.

Two components turn out not to matter, and we report this rather than hide it. Removing Pareto pruning reproduces XSLA's outcome exactly — 65.16 DCAY to five significant figures — which is the empirical confirmation Lemma 1 predicts; pruning buys enumeration cost, not decision quality. Removing online re-calibration changes DCAY by 0.01. We had expected it to matter; Section 6.9 explains why it does not.

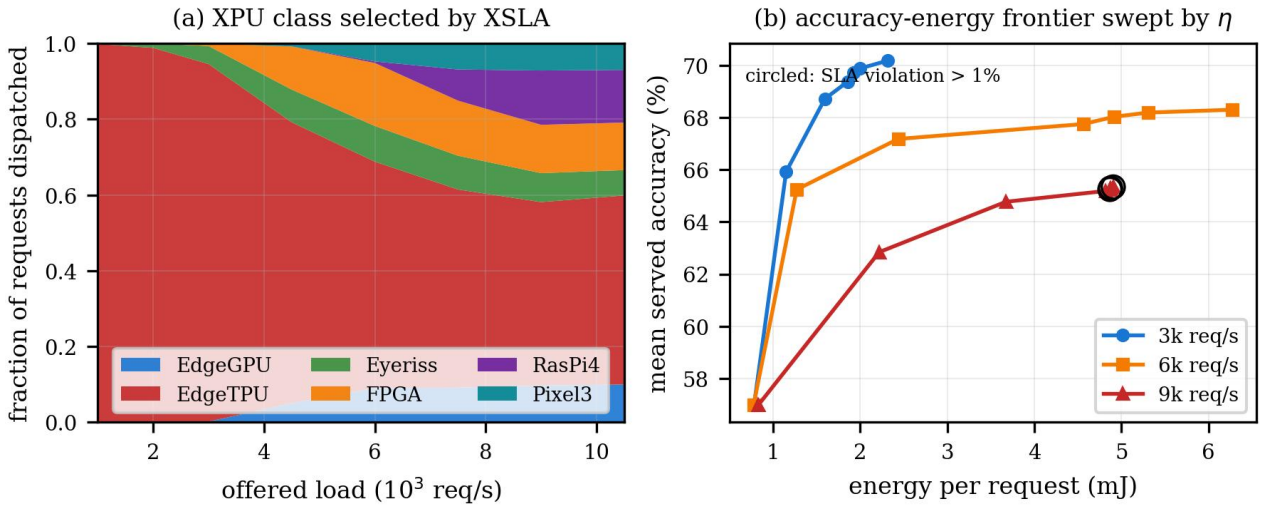

Figure 4 (a) Fraction of Requests XSLA Dispatches to Each XPU Class as Offered Load Rises; (b) Accuracy–Energy Frontier Traced by the Quality Weight η

Note: (a) The scheduler discovers a spill-over order from Edge TPU to FPGA/Eyeriss to Edge GPU/Pixel 3 and finally to the Raspberry Pi 4. (b) Circled points violate more than 1% of deadlines and mark the safe operating limit.

Table 4 Component Ablation at 9,000 req/s

Configuration	DCAY	Δ DCAY	SLA viol.	Mean acc.	mJ/req	p99 (ms)	Candidates
XSLA (full)	65.16	—	0.0%	65.19	4.82	32.7	75.4
no variant switching	26.45	−59.4%	62.3%	67.28	10.29	2097.4	12.0
static quality weight	47.53	−27.1%	28.0%	65.53	4.95	205.9	75.4
FIFO queues (class-agnostic)	64.01	−1.8%	0.0%	64.01	4.67	7.1	75.4
no online re-calibration	65.17	+0.0%	0.0%	65.19	4.81	33.2	75.4
no Pareto pruning	65.16	±0.0%	0.0%	65.19	4.82	32.7	96.0

Note: "Candidates" is the mean number of (variant, XPU) pairs scored per dispatch.

6.6 Robustness to the Quality Weight

Figure 5(a) sweeps V0 over six octaves with adaptation on (κ = 20) and off (κ = 0). Static weighting is brittle: mean DCAY spans 15.7 points across the range and falls below 50 for V0 ≥ 0.4, because a large static weight over-commits at high load. Adaptation compresses the spread to 10.0 points, a 1.6x reduction, and keeps DCAY above 61 for every V0 in [0.025, 0.8]. At our operating point V0 = 0.1, adaptation is worth 5.8 DCAY points (67.43 vs 61.68). The practical value of the adaptive weight is therefore as much that it removes the need for precise tuning as that it improves the optimum.

6.7 Pruning Benefit and Dispatch Cost

Figure 5(b) quantifies Lemma 1 honestly. Because our main dispatcher scores candidates as masked fixed-shape arrays, pruning changes the candidate count but not the vectorised work, so it shows no wall-clock gain there. We therefore also measured an explicit enumerating dispatcher of the kind a production implementation would use. Pruning removes 24.0% of pairs for the default 8-variant zoo and 57.8% for a 32-variant zoo, giving enumeration speed-ups of 1.27x and 2.04x. The benefit grows with zoo size because the Pareto frontier grows sub-linearly in the number of variants: 45.8 surviving pairs at 4 variants per task, 172.8 at 32.

Absolute dispatch cost is modest. In unoptimised Python, XSLA takes 29.2 µs per decision at K = 12 and 36.6 µs at K = 96 — an 8x larger pool for a 25% cost increase, because the per-accelerator work is vectorised. Scaling the zoo from 4 to 32 variants per task raises cost from 29.1 µs to 32.5 µs. Against a 10 ms latency-critical budget the overhead is 0.3%.

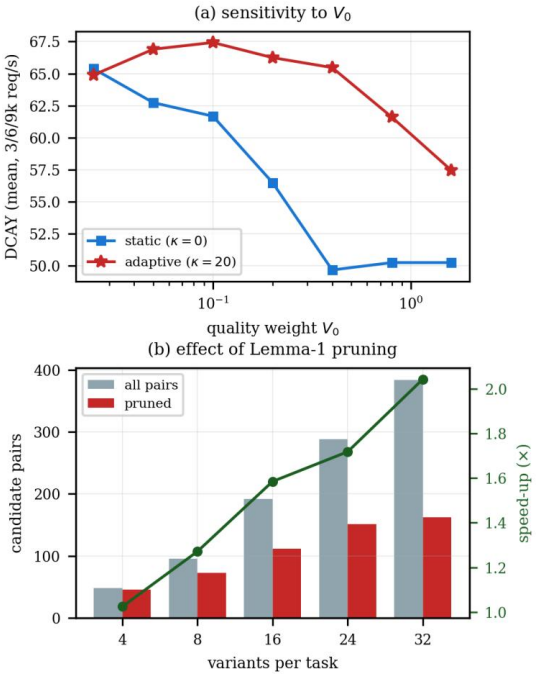

Figure 5 (a) Sensitivity to the Quality Weight V0 under a Static (κ = 0) and a Load-Adaptive (κ = 20) Rule; (b) Candidate Pairs before and after Lemma-1 Pruning as the Zoo Grows, and the Resulting Enumeration Speed-Up (Right Axis)

Note: (a) Adaptation compresses the DCAY spread from 15.7 to 10.0 points.

6.8 Bursty and Non-Stationary Arrivals

Figure 6 replaces Poisson arrivals with a two-state MMPP (4x rate in the burst state) and a diurnal process with ±85% amplitude. Under diurnal load XSLA behaves as under Poisson: 0.0% violations at 3,000 and 6,000 req/s and 1.4% at 9,000, DCAY 68.52/66.16/62.07, against 66.34/38.86/30.85 for Accuracy-Greedy and 19.88/1.17/0.74 for EDF+EFT.

MMPP is the harder test, because instantaneous burst rates exceed pool capacity and some deadline misses are unavoidable for any policy. Here the class-aware design earns its place: XSLA keeps latency-critical violations at 0.0% at 3,000 and 6,000 req/s and 3.2% at 9,000, while shedding best-effort work (23.3% BE violations at 6,000 req/s). Every baseline fails both classes together — Accuracy-Greedy 25.9% LC, EDF+EFT 28.4% LC, Latency-Greedy 24.5% LC, Energy-Greedy 24.2% LC at 6,000 req/s. XSLA's aggregate DCAY of 57.95 at that load still beats the best baseline (Accuracy-Greedy, 52.42) by 10.6%. Under overload the framework thus trades the class the SLA says is expendable, which is the behaviour an operator contracts for and which no baseline reproduces.

6.9 Unmodelled Device Slowdown

We impose a 1.6x systematic slowdown on both Edge GPUs and one FPGA, unknown to the scheduler a priori, representing thermal throttling or a co-runner. At 6,000 req/s XSLA still records 0.0% violations and DCAY 67.13, a 0.9% loss relative to the unperturbed 67.75. Accuracy-Greedy loses 19.3% (51.57 → 41.63) and its violations rise from 24.2% to 40.6%; EDF+EFT is unchanged only because it had already collapsed.

This experiment also explains the null result for online re-calibration: the variant with re-calibration disabled scores 67.48, marginally above the full system. The reason is structural. A slowed accelerator drains its queue more slowly, so its measured backlog W(k) rises, and the feasibility test and drift term see the slowdown through the queue within one service time. Explicit latency re-calibration corrects only the newly dispatched request's own estimate, a second-order effect. We conclude that in a queue-driven dispatcher, backlog feedback subsumes most of the value of latency-model correction — a useful simplification for implementers, and a caution against assuming profiling machinery is necessary.

6.10 Value of XPU Heterogeneity

Table 5 holds the pool at 12 accelerators and varies its composition at 3,000 req/s. Removing the NPU class costs XSLA 7.2% of DCAY and multiplies energy by 4.7x; a homogeneous Edge GPU pool costs 10.5% of DCAY and 5.2x energy. The more revealing column is the baseline: Accuracy-Greedy, which is optimal on the full pool, fails outright on both NPU-free pools (43.3% and 44.4% violations, DCAY ≈ 39), because without the Edge TPU's headroom its per-request greed exceeds capacity. XSLA remains at 0.0% violations on every composition. The framework's value therefore grows as the pool becomes more constrained and more heterogeneous, which is the direction real edge deployments take.

A homogeneous NPU pool is a boundary case: it gives the lowest energy of all (1.23 mJ) and Accuracy-Greedy attains the ceiling, because 12 Edge TPUs at 3,000 req/s are barely loaded. Heterogeneity is not intrinsically valuable; it is valuable when it is the only way to add capacity without adding cost — exactly when scheduling becomes hard.

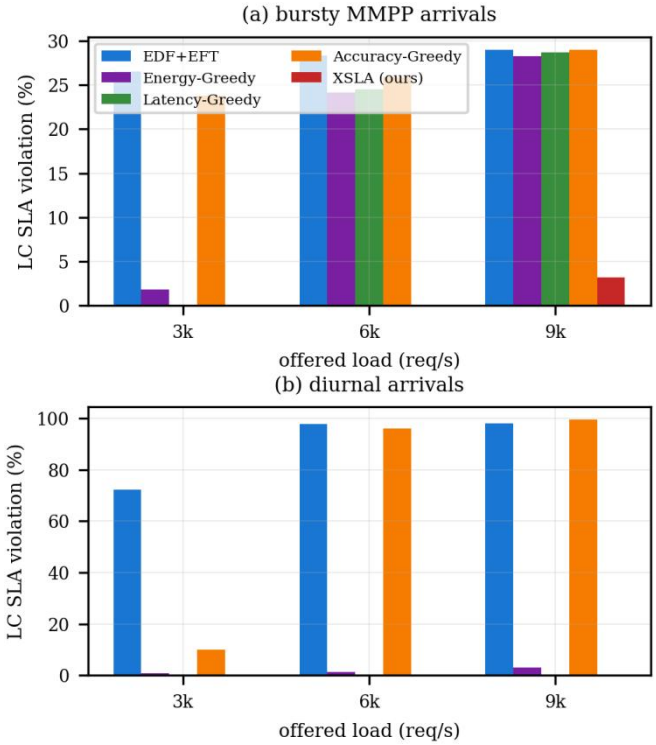

Figure 6 Latency-Critical SLA Violation under (a) Bursty MMPP and (b) Diurnal Arrivals

Note: XSLA is the only policy that isolates the latency-critical class from bursts; it does so by shedding best-effort work rather than degrading both classes together.

Table 5 Pool Composition at 3,000 req/s, Always 12 Accelerators

Pool	XSLA DCAY	XSLA mJ	AccG. DCAY	AccG. viol.
A: all 6 classes	69.33	1.86	70.48	0.0%
B: no NPU	64.31	8.78	39.37	43.3%
C: no ASIC/FPGA	69.37	1.86	70.48	0.0%
D: GPU + NPU	69.37	1.86	70.48	0.0%
E: NPU only	67.72	1.23	70.48	0.0%
F: GPU only	62.08	9.59	39.03	44.4%

Note: XSLA never violates an SLA on any composition; Accuracy-Greedy fails outright on both NPU-free pools. Latency-Greedy is load-insensitive at 56.94 DCAY (58.25 on pool F).

6.11 SLA Tightness

Tightening the latency-critical deadline from 25 ms to 4 ms at 6,000 req/s reduces XSLA's DCAY only from 67.94 to 66.09 (−2.7%), with 0.0% violations throughout, whereas Accuracy-Greedy is insensitive because already saturated (52.12 → 51.06 DCAY, 57.7% → 61.4% LC violations). XSLA absorbs a 6x tightening by shifting to faster variants and accelerators — the intended use of the $\hat{C}/d$ term.

6.12 A Narrow-Spread Zoo with Support Gaps

To test the framework where its central lever is weakest, we build a second zoo from each task's eight most accurate architectures regardless of Edge TPU support. Mean XPU support drops to 5.9 of 6, so coverage gaps become active constraints, and the within-task accuracy spread narrows from 10.8 points to 1.9. XSLA sustains 0.0% violations to 6,000 req/s with DCAY 69.62/69.09/68.17 at 3,000/4,500/6,000 req/s, against 70.48/56.18/29.92 for Accuracy-Greedy and 67.64/67.56/67.52 for Latency-Greedy. The margin over Latency-Greedy shrinks to 1.0%, exactly as expected: when variants are nearly equally accurate there is little for accuracy-aware co-selection to win and the correct behaviour is to converge on the latency-greedy choice. That XSLA converges follows from the absolute normalisation of Section 5.4; per-zoo min–max normalisation instead over-valued 0.2-point differences and cost 14.4% deadline misses. Above 6,000 req/s this zoo exceeds pool capacity for every policy, so we do not report further.

6.13 Sensitivity to the Energy Estimates

Varying the three estimated power constants of Table 1 by ±30% moves XSLA's DCAY by 0.04% (67.70/67.72/67.73) with violations at 0.0%; energy per request moves proportionally, as it must (4.14/4.58/4.97 mJ). Because decisions depend on energy only through a normalised term weighted by Vt, uniformly scaling three of six platforms does not reorder them. No conclusion here rests on those constants.

7 DISCUSSION AND LIMITATIONS

The results support a simple thesis: on a heterogeneous edge pool, accuracy is a scheduling resource. Every policy that fixes the model saturates at 3,000 req/s; every policy that maximises accuracy per request in isolation destabilises the pool; the policy that spends accuracy as load rises serves 3.5x more traffic within its SLA. What makes this affordable is that the accuracy–cost frontier is tiny (O3) and prunes losslessly per accelerator (Lemma 1).

Several limitations bound the claims. (i) The evaluation is simulation-based: service times, energies and accuracies are real measurements, but queueing, dispatch and contention are modelled, so a prototype would add loading, DMA and framework overheads we do not capture. (ii) We model one inference per accelerator and no batching — faithful to fixed-function NPUs, ASICs and FPGAs but pessimistic for the Edge GPU, where batching trades latency for throughput [9-10]; folding batch formation into the co-selection score is the obvious extension. (iii) We do not model DVFS, so joint variant–accelerator–frequency selection remains open. (iv) Arrival processes are synthetic. (v) Accuracies come from a 12-epoch budget and energies for three platforms from a P·t model; Sections 3.1 and 6.13 bound the consequences. (vi) The framework schedules within one site and does not consider offloading [1-2, 14].

Two findings ran against our expectations and may save others effort. Online latency re-calibration is nearly worthless in a queue-driven dispatcher, because backlog already encodes device slowdown. And normalising accuracy by the zoo's own range — the obvious choice — is harmful: it makes the scheduler as aggressive in a zoo whose variants differ by 2 points as in one where they differ by 17. Absolute normalisation is required for the framework to transfer across zoos.

8 CONCLUSION

We presented XSLA, an SLA-aware scheduling framework for edge AI inference on heterogeneous XPU pools that treats model-variant selection and accelerator selection as one online decision. Its core components are a provably lossless per-accelerator Pareto pruning test, a drift-plus-penalty dispatch rule with a load-adaptive quality weight, and class-aware backlog estimation consistent with the priority discipline of the device queues. Evaluated on a testbed assembled from real measurements of 6 accelerator classes and 24 model variants, XSLA holds SLA violation at zero up to 9,000 req/s where the strongest comparable policy misses 47% of deadlines, improves deadline-constrained accuracy yield by 14.4–81.4% at equal load, and cuts energy per request by 37–63%. Its advantage grows as the pool becomes more constrained, and it degrades to the correct limiting behaviour when the deployed zoo offers little accuracy to trade. Code, the derived model zoo and all experiment outputs accompany this paper.

COMPETING INTERESTS

The authors have no relevant financial or non-financial interests to disclose.

REFERENCES

[1] Zhou Z. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. Proceedings of the IEEE, 2019, 107(8): 1738-1762.
[2] Kang Y. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. In: Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2017: 615-629.
[3] Romero F. INFaaS: Automated model-less inference serving. In: Proceedings of USENIX ATC, 2021: 397-411.
[4] Narayanan D. Heterogeneity-aware cluster scheduling policies for deep learning workloads. In: Proceedings of USENIX OSDI, 2020: 481-498.
[5] Li C. HW-NAS-Bench: Hardware-aware neural architecture search benchmark. In: Proceedings of the International Conference on Learning Representations (ICLR), 2021.
[6] Dong X, Yang Y. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In: Proceedings of the International Conference on Learning Representations (ICLR), 2020.
[7] Crankshaw D. Clipper: A low-latency online prediction serving system. In: Proceedings of USENIX NSDI, 2017: 613-627.
[8] Gujarati A. Serving DNNs like clockwork: Performance predictability from the bottom up. In: Proceedings of USENIX OSDI, 2020: 443-462.
[9] Shen H. Nexus: A GPU cluster engine for accelerating DNN-based video analysis. In: Proceedings of ACM SOSP, 2019: 322-337.

[10] Zhang C. MArk: Exploiting cloud services for cost-effective, SLO-aware machine learning inference serving. In: Proceedings of USENIX ATC, 2019: 1049-1062.
[11] Zhang H. SHEPHERD: Serving DNNs in the wild. In: Proceedings of USENIX NSDI, 2023: 787-808.
[12] Matsubara Y, Levorato M, Restuccia F. Split computing and early exiting for deep learning applications: Survey and research challenges. ACM Computing Surveys, 2023, 55(5): 1-30.
[13] Ran X. DeepDecision: A mobile deep learning framework for edge video analytics. In: Proceedings of IEEE INFOCOM, 2018: 1421-1429.
[14] Nigade V. Jellyfish: Timely inference serving for dynamic edge networks. In: Proceedings of IEEE RTSS, 2022: 277-290.
[15] Jeong J S. Band: Coordinated multi-DNN inference on heterogeneous mobile processors. In: Proceedings of ACM MobiSys, 2022: 235-247.
[16] Jia F. CoDL: Efficient CPU-GPU co-execution for deep learning inference on mobile devices. In: Proceedings of ACM MobiSys, 2022: 209-221.
[17] Ling N. RT-mDL: Supporting real-time mixed deep learning tasks on edge platforms. In: Proceedings of ACM SenSys, 2021: 1-14.
[18] Kim S. DREAM: A dynamic scheduler for dynamic real-time multi-model ML workloads. In: Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2023: 73-86.
[19] Fan H. Sparse-DySta: Sparsity-aware dynamic and static scheduling for sparse multi-DNN workloads. In: Proceedings of IEEE/ACM MICRO, 2023: 353-366.
[20] Weng Q. MLaaS in the wild: Workload analysis and scheduling in large-scale heterogeneous GPU clusters. In: Proceedings of USENIX NSDI, 2022: 945-960.
[21] Liu C L, Layland J W. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM, 1973, 20(1): 46-61.
[22] Topcuoglu H, Hariri S, Wu M-Y. Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems, 2002, 13(3): 260-274.
[23] Neely M J. Stochastic Network Optimization with Application to Communication and Queueing Systems. Morgan & Claypool, 2010.
[24] Chen Y-H. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE Journal of Solid-State Circuits, 2017, 52(1): 127-138.
[25] Zhang L L. nn-Meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices. In: Proceedings of ACM MobiSys, 2021: 81-93.
[26] Reddi V J. MLPerf inference benchmark. In: Proceedings of ACM/IEEE ISCA, 2020: 446-459.
[27] Sandler M. MobileNetV2: Inverted residuals and linear bottlenecks. In: Proceedings of IEEE CVPR, 2018: 4510-4520.
[28] Tan M, Le Q V. EfficientNet: Rethinking model scaling for convolutional neural networks. In: Proceedings of the International Conference on Machine Learning (ICML), 2019: 6105-6114.
[29] Cai H. Once-for-All: Train one network and specialize it for efficient deployment. In: Proceedings of the International Conference on Learning Representations (ICLR), 2020.
[30] Shahrad M. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In: Proceedings of USENIX ATC, 2020: 205-218.