Journal of Computing and Electronic Information Management
ISSN: 2413-1660 | Vol. 23, No. 2, 2026

Hardware-Aware Adaptive Quantization for Low-Latency Edge AI Inference: Measurement-Guided Layer Selection

Hannah Whitfield [1], Oliver Bennett [2], Rebecca Lloyd [1, *]

[1] Leeds University Business School, University of Leeds, Leeds, United Kingdom
[2] Warwick Business School, University of Warwick, Coventry, United Kingdom

[*] **Corresponding author**: rebecca.lloyd.paper06@gmail.com

Abstract: Uniform low-precision conversion reduces weight storage but need not minimize CPU inference latency. We propose a hardware-aware precision-selection procedure that profiles complete neural networks with layer wise FP32 or dynamic INT8 operators and chooses the fastest mask satisfying a validation-accuracy constraint. The method exhaustively evaluates eight precision masks for a three-linear-layer classifier and requires no quantization-aware retraining. Using the 1,797-sample handwritten-digits dataset, three hidden widths (256, 512, 1,024), three initialization seeds, and batch sizes 1 and 16 yields 144 measured configurations. The smallest single-request case retains FP32; intermediate matrix sizes favor selective INT8 on the dominant hidden layer; the widest batch-16 model favors uniform INT8. An independent timing pass with frozen masks and held-out inputs confirms mean speedups of 1.87× at width 512/batch 16 and 4.12× at width 1,024/batch 16, relative to FP32. Selected masks match the paired FP32 held-out test accuracy in all configurations. The results show that small-scale, backend-aware precision profiling can improve CPU inference without optimizing quantization kernels or retraining the deployed model.

Keywords: Edge AI; Mixed precision; Hardware-aware quantization; Dynamic INT8; CPU latency; Deployment optimization.

1. Introduction

Deploying deep neural networks under strict latency and storage budgets requires optimization decisions that are consistent with the target execution environment. Model compression has reduced storage and memory traffic [1], while integer arithmetic and post-training quantization provide efficient numerical representations [2], [3]. Nevertheless, theoretical bit-operation counts alone cannot predict measured inference time: kernel dispatch, activation conversion, memory layout, and workload shape all affect execution. This distinction is particularly important for edge-oriented models, where a few microseconds can form a sizable fraction of the total request latency [4], [5].

Hardware-aware design has become central to mobile networks and embedded inference. MobileNet and its successors redesign operators for portable devices [6]–[8], ShuffleNet V2 explicitly studies direct versus indirect efficiency metrics [9], and compact architectures such as SqueezeNet and MCUNet target stringent resource limits [10], [11]. For an existing trained model, however, changing precision at selected layers can be simpler than retraining or searching for a new architecture.

This paper studies deployment-time adaptive precision assignment, not per-input dynamic switching. A three-layer multilayer perceptron is treated as a set of independently convertible linear operators, each executable in either floating-point or dynamic INT8 precision. Instead of assigning INT8 everywhere, the proposed selector times complete end-to-end inference on the available CPU backend and chooses the fastest configuration that satisfies an accuracy tolerance on a validation set. The practical emphasis is a transparent, low-computation design with measured inference, accuracy, and serialized-state outcomes.

The study makes three contributions: (i) a direct-latency-constrained FP32/INT8 assignment rule requiring no retraining after conversion; (ii) a complete eight-mask comparison across three network widths and two batch sizes; and (iii) a quantitative demonstration that the best precision mask changes with operator size and batch regime, including instances in which no quantization is the correct latency choice.

2. Related work

2.1. A. Neural-Network Quantization

Quantization-aware training methods optimize numerical behavior during learning. PACT adjusts activation clipping [12], learned step-size quantization optimizes quantizer resolution [13], and DoReFa-Net, XNOR-Net, quantized neural networks, and BinaryConnect explore aggressively reduced numerical precision [14]–[17]. These works establish compression mechanisms, but they are not intended as measurements of Python-level dynamic INT8 operator latency in an existing CPU execution stack.

Post-training methods avoid full retraining. Banner et al. address low-bit conversion [18]; data-free equalization and adaptive rounding mitigate conversion error [19], [20]; BRECQ and QDrop improve reconstruction quality [21], [22]; and ZeroQ considers limited access to calibration data [23]. The present method uses standard backend-supplied INT8 kernels and focuses on where conversion should be applied to achieve the lowest measured latency, rather than introducing a different quantizer.

2.2. B. Hardware-Aware Optimization

HAQ uses hardware feedback for mixed precision [24], while HAWQ, HAWQ-V2, and HAWQV3 exploit sensitivity information and execution constraints [25]–[27]. AMC and APQ integrate compression decisions with architecture or

policy search [28], [29]. NetAdapt, FBNet, MnasNet, ProxylessNAS, and once-for-all networks further demonstrate platform-dependent efficiency decisions [30]–[34]. Compared with those broader search systems, exhaustive evaluation of three binary layer choices is computationally lightweight and sufficient to isolate the latency effect of selective INT8 conversion.

Compiler systems such as TVM, Ansor, and Glow optimize lower-level tensor execution [35]–[37]. Recent large-model approaches including GPTQ, LLM.int8(), SmoothQuant, and AWQ extend quantization toward large transformer workloads [38]–[41]. This work targets a contrasting small CPU classification workload, implemented in PyTorch [42] with a scikit-learn dataset [43]. Adam [44] is used to train the FP32 source models; the 8×8 dataset is documented by scikit-learn [45].

3. Methodology

3.1. Precision Search Space

Let f(x; W) be a trained neural network with L linear layers. A binary vector $z = (z_1,\ldots,z_L)$ defines deployment precision: $z\ell = 0$ keeps layer ℓ in FP32 and $z\ell = 1$ replaces it with dynamic INT8. In the evaluated network L = 3, so the candidate set comprises $2^3 = 8$ masks from 000 to 111, written in forward order (first hidden, second hidden, output). The output of a quantized linear operator is returned to floating-point form by the selected backend; ReLU remains a standard FP32 operation. Thus the configuration is mixed-precision at the operator level, rather than a fully integer-only graph.

Dynamic INT8 quantizes weights ahead of inference and transforms activations as needed at execution time. For narrow matrices, the extra packing, activation scaling, and dispatch work can exceed the benefit of integer arithmetic. The relevant objective is therefore measured end-to-end latency T(z; h, n), conditioned on hardware/software backend h and batch size n, not just model byte count or approximate multiply-accumulate count.

3.2. Validation-Constrained Selection

Let Aval(z) be validation classification accuracy in percentage points and let $\hat{T}(z; h, n)$ denote the median inference time from repeated measurements. The deployment decision is defined as

$z^* = \arg\min \hat{T}(z; h, n)$, subject to $Aval(z) \geq Aval(000) - \delta$.

Here $\delta = 0.5$ percentage points is fixed before benchmarking. If multiple feasible masks have identical measured latency, smaller serialized state and then lexical mask order break ties. The FP32 mask is always feasible, so a no-quantization alternative remains available when INT8 is slower in profiling. Since L = 3, exact enumeration is inexpensive; larger architectures could replace enumeration with a beam or greedy candidate generator, while retaining the same measurement and accuracy test.

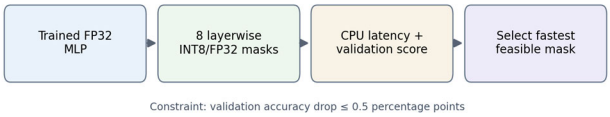


Fig. 1 Deployment-time hardware-aware precision selection. The latency measurements are taken on the execution backend being optimized; no quantization-aware fine-tuning is performed.

3.3. Timing and Storage Measurement

For every candidate, inference is warmed up for 100 calls; then 5 blocks of 220 individual calls are timed using perf_counter_ns in inference mode with tensor creation outside the timed region. The reported scalar is the median over 1,100 timed calls per configuration. Timings include dynamic quantization overhead, layer invocation, and activation functions. A serialized PyTorch state dictionary provides a deployment-relevant model-storage measure (including container metadata), rather than assuming an ideal fourfold reduction for all masks. Exactly one CPU intra-op thread and one inter-op thread are used.

3.4. Search Procedure and Computational Cost

The policy can be expressed as a small deployment pipeline. First, select one FP32 checkpoint using its validation set. Second, instantiate all layerwise precision masks without modifying the FP32 source weights. Third, calculate validation accuracy and serialized state size for each mask. Fourth, benchmark whole-model inference at the intended batch size with fixed tensors and CPU thread count. Finally, return the feasible mask with minimum median latency. For the three-linear-layer architecture, the conversion/evaluation count is eight per source checkpoint and batch size; no differentiable policy network, Hessian approximation, or reinforcement-learning controller is required. The timing measurement is repeated when batch size or execution backend changes.

A useful feature of this objective is that it does not require an additive latency assumption, $T(z) \approx \Sigma\ell T\ell(z\ell)$. Such a sum may omit interface conversion overhead and dispatcher effects. Direct complete-model timing is inexpensive in the present eight-mask search and avoids imposing that approximation. The same accuracy threshold applies to every mask, preventing a fast but substantially degraded model from being selected solely because of throughput.

4. Experimental design

4.1. Dataset, Network and Training

The public scikit-learn digits dataset contains 1,797 grayscale digit images of 8×8 pixels and 10 classes [43], [45]. Pixel values in [0,16] are divided by 16. Stratified splits produce 1,077 training, 360 validation, and 360 test instances, with fixed split seed 42. The test labels are not used to train models or select precision masks; timing inputs are sampled from the validation split. Separate source models are trained for random seeds 0, 1, and 2.

The MLP is 64–W–W–10 with ReLU between its three linear layers and $W \in \{256, 512, 1024\}$. FP32 networks use cross-entropy loss and Adam with learning rate 0.001, batch size 64, and 28 epochs. For each random seed, the checkpoint with the highest validation accuracy (cross-entropy as a tie breaker) is used for all eight conversions. Therefore, configuration comparisons hold the learned FP32 weights and data partitions fixed. No quantization-aware training or post-conversion update occurs.

4.2. Measured Processor and Evaluation Rules

Experiments were conducted in a Linux x86-64 virtualized environment with an AMD EPYC 9V74 CPU identifier, PyTorch 2.10.0+cpu and the x86 quantized backend. The process used a single execution thread for all timing results. We report batches of 1 and 16 to capture latency-sensitive

single-request execution and modest request aggregation. Figures and tables summarize mean values across three trained seeds; speedup is calculated for each seed as TFP32/Tselected before aggregation. Absolute times characterize the identified x86 backend and do not imply measured latency on ARM microcontrollers or a physical edge board.

Table I. Hardware-aware masks and mean median latency

W	Batch	Selected mask	FP32 (µs)	Selected (µs)	Speedup
256	1	000	24.3	24.3	1.00×
256	16	010	55.8	54.1	1.03×
512	1	010	45.4	44.1	1.03×
512	16	110,010,010	155.1	84.9	1.83×
1024	1	010	99.3	63.0	1.58×
1024	16	111	684.6	163.7	4.18×

4.3. Evaluation Protocol and Reproducibility

Each width has three independently initialized models trained on exactly the same stratified partitions. This paired design makes each precision comparison relative to its own FP32 source checkpoint, rather than comparing separately trained floating-point and quantized networks. For a given width and batch, all masks use the same validation-input tensor shape and the same input-generation seed. Runtime calls execute under torch.inference_mode() and are preceded by operator warm-up. Latency is summarized by the median of individual calls rather than by a single aggregate pass, and the data release contains the underlying mask-level medians and 95th-percentile observations [46-51].

The performance records distinguish three quantities: per-call time in microseconds, per-seed speedup relative to the corresponding FP32 execution, and storage bytes from the serialized state dictionary. Reported aggregate speedups are means of per-seed speedup ratios, rather than quotients of pooled unpaired times. This matters when comparing independent training runs and when the latency distribution changes across the 256-, 512-, and 1024-unit models. The model-selection process uses validation accuracy and measured validation-input latency; held-out test labels are only used for final accuracy reporting [52-55].

4.4. Independent Latency-Confirmation Protocol

To separate precision-mask search from final latency assessment, the chosen mask for each width, seed, and batch is frozen using the first profiling pass. In a separate execution pass, the corresponding FP32 networks are retrained with the same fixed seeds and training recipe, and each frozen mask is compared with FP32 (000) and uniform INT8 (111). New input tensors are drawn from the held-out test split only for timing; test labels do not enter precision selection. The confirmation run uses 80 warm-up inferences, five blocks of 100 individual calls per distinct mask, and randomized mask execution order within each block. Its reported latency is the median of the five block medians, with speedup computed relative to the paired FP32 model. This independent measurement addresses the optimistic bias of reporting the minimum of noisy search-time measurements as final performance.

5. Results and discussion

5.1. Batch-Dependent Latency

Table I shows that hardware-aware selection is conservative for small models but increasingly valuable as matrix dimensions grow. At W = 256 and batch 1, every INT8-containing mask introduces enough overhead that the fastest eligible configuration is 000 for all three seeds. At batch 16, selective quantization of the second hidden layer (010) yields a mean 1.03× speedup. W = 512 favors 010 for batch 1 (1.03×) and usually 010 for batch 16, reaching 1.83× mean speedup. For W = 1024, 010 reaches 1.58× at batch 1, while all-INT8 (111) becomes fastest at batch 16, reaching 4.18×.

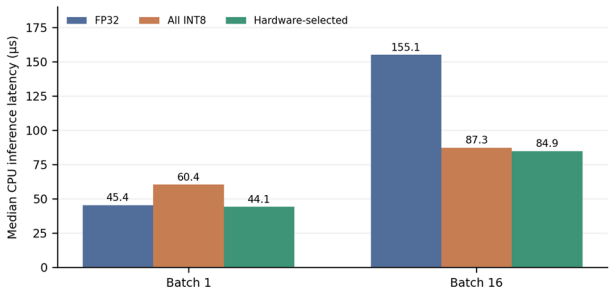


Fig. 2 Mean measured median CPU inference time for W = 512 (three training seeds). Lower is better. Fully quantized execution is slower than FP32 at batch 1, whereas selective quantization improves latency.

5.2. Full-Precision Versus Blanket INT8

Uniform INT8 and hardware-selected INT8 should not be treated as interchangeable baselines. For W = 256, batch 1, blanket INT8 achieves only 0.43× FP32 throughput-equivalent speedup (i.e., it is slower), whereas selection retains exactly the FP32 latency. For W = 512, batch 1, full INT8 is also slower than FP32 (0.75× speedup), while mask 010 improves latency. At W = 1024, the main intermediate matrix is large enough for INT8 computation to be beneficial; nevertheless, adding quantized first and output layers still reduces the single-request speedup relative to mask 010. At W = 1024 and batch 16, those extra conversion costs are amortized and full INT8 becomes competitive with every mixed mask.

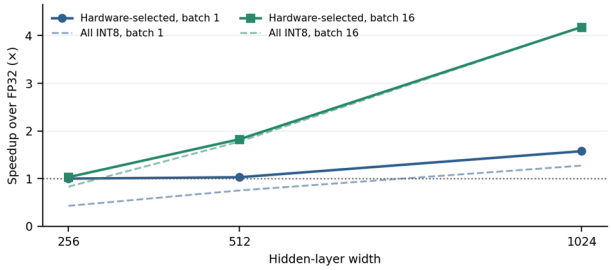


Fig. 3 Speedup relative to FP32 across hidden widths. Solid lines use validation-constrained hardware selection; dashed lines use blanket INT8. Each point is the mean of three per-seed speedup ratios.

5.3. Layer-Level Ablation

Table II isolates the effect of each quantized layer. Mask 001 quantizes only the 512-to-10 output projection, and mask 100 quantizes only the 64-to-512 input projection; both are slower than FP32. By contrast, mask 010 targets the 512-to-

512 matrix and gives the principal latency gain while preserving the FP32 result for every seed at this width. Quantizing additional layers (011, 110, 111) can reduce bytes further, but these choices also pay extra runtime conversion overhead. The observed trade-off motivates hardware profiling at the complete-graph level rather than extrapolation from isolated weight size.

Table II. Eight-precision ablation at w = 512

Mask	B=1 (µs)	B=16 (µs)	Test acc. %	Size (KiB)
000	45.4	155.1	97.31	1179
001	66.8	180.0	97.31	1164
010	44.1	85.1	97.31	411
011	55.7	93.9	97.50	397
100	65.2	181.7	97.31	1083
101	77.3	185.4	97.31	1069
110	52.3	85.2	97.31	316
111	60.4	87.3	97.31	302

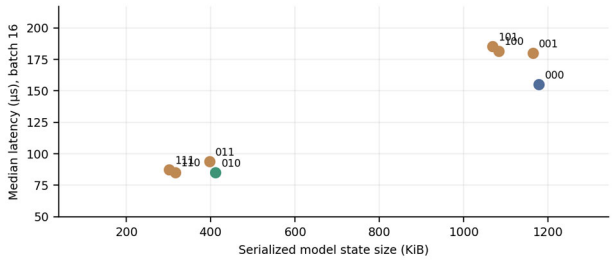


Fig. 4 Model-state size versus mean latency for all eight masks at W = 512, batch 16. Labels indicate the quantized-layer bit mask (e.g., 010 denotes INT8 only in the middle layer).

5.4. Accuracy and Model Footprint

The selected mask matches the corresponding FP32 model's held-out test accuracy in every width–seed–batch run. Mean test accuracies for W = 256, 512, and 1024 are 96.67%, 97.31%, and 97.50%, respectively. The unchanged test scores are observed outcomes, not guarantees that all quantizers are lossless. Since the selected weight configuration is batch-dependent, storage savings are reported by runtime regime rather than as a single global property.

Table III. Test accuracy and serialized-state reduction

W	Batch	Size saved %	FP32 acc. %	Sel. acc. %	All-INT8 speedup
256	1	0.0	96.67	96.67	0.43×
256	16	57.2	96.67	96.67	0.83×
512	1	65.1	97.31	97.31	0.75×
512	16	67.8	97.31	97.31	1.78×
1024	1	69.8	97.50	97.50	1.27×
1024	16	74.8	97.50	97.50	4.18×

For batch 1, the deployment selector reduces the model state size by 65.1% at W = 512 and 69.8% at W = 1024 while improving median latency. At W = 256 it deliberately gives up the storage benefit to protect latency. At batch 16, the average saved storage is 57.2%, 67.8%, and 74.8% across W = 256, 512, and 1024, respectively. This is a storage/latency/accuracy selection result, not a generic claim that smaller serialized files always execute faster.

5.5. Stability Across Training Seeds

The principal mask choices are consistent across random initializations. For W = 256, all three batch-1 experiments choose 000 and all three batch-16 experiments choose 010. At W = 512, every batch-1 run chooses 010; at batch 16, two runs choose 010 and one chooses 110. At W = 1024, all three batch-1 runs choose 010 while all three batch-16 runs choose 111. Consequently, changes in precision assignment are primarily associated with the matrix and batch shape rather than an idiosyncratic accuracy checkpoint. The exceptional W = 512, batch-16 choice shows that near-tied masks can be reordered by small runtime fluctuations and should not be interpreted as fundamentally different design regimes.

Across the three seeds, the standard deviation of the selected speedup is 0.03× or less for both batch sizes at W = 512, while W = 1024 and batch 16 shows a larger 0.17× spread around the 4.18× mean. These are descriptive cross-seed variations on one virtualized CPU measurement environment, not cross-device confidence intervals. They reinforce the value of evaluating the precision policy on the backend and batch mix that will actually execute the deployed workload [56-60].

5.6. Interpretation and Deployment Scope

The measurements indicate two operational regimes. For small batch-1 matrices, kernel startup and dynamic activation quantization can overwhelm lower-precision arithmetic; a hardware-aware procedure must be permitted to select FP32. For sufficiently large hidden matrices or moderately batched inference, lower-precision matrix multiplication offsets these costs. Furthermore, the best assignment is not necessarily monotone in the number of INT8 layers: at W = 512, a single well-chosen INT8 layer is faster than a larger count of quantized layers in the single-request setting.

The proposed search is intentionally specific to the model, backend, and batch distribution at deployment. A re-profile is appropriate if the CPU, BLAS/quantized kernel implementation, threading policy, or expected batch changes. The absolute figures are for a virtualized AMD x86 processor and a compact 8×8 image classification task; they establish the kernel-selection mechanism but do not constitute mobile-device energy measurements or a benchmark of modern high-resolution vision networks. The measured gain comes from deployment-time policy selection rather than a new low-bit arithmetic kernel.

5.7. Engineering Implications

Three design consequences follow from the observed configurations. First, activation quantization should be treated as a real computational cost. In dynamic INT8, inputs to a quantized linear layer are converted at runtime; this is particularly visible when the operator has only a small number of multiply-accumulates. Second, model compression and latency optimization are not identical objectives. At W = 512, batch 1, a mask with fewer stored bytes is not automatically faster because different layer interfaces require different dispatch and conversion paths. Third, hardware-aware configuration need not involve an expensive search algorithm when the precision space is small. The complete eight-mask experiment directly checks the relevant alternatives.

At deployment, a system serving mostly individual requests could save mask 010 for W = 512 or W = 1024, whereas a system consistently batching 16 requests could prefer a different mask. This is a configuration-time decision: the experiment does not switch the numerical format on individual samples and does not include a runtime gate or an

uncertainty detector. If the workload batch distribution changes, the measured decision table can be regenerated without retraining the underlying FP32 model. This makes the procedure suitable as a small software-engineering layer between model training and CPU inference deployment.

5.8. Quantization Effects Are Not Simply Additive

The full eight-mask ablation also permits a direct test of a tempting but inaccurate simplification: predicting whole-network latency from separately quantized layers. Define $\Delta\ell = T(e\ell) - T(000)$, where $e\ell$ is the mask with only layer ℓ quantized. Under a fully additive model, the predicted all-INT8 latency would be $Tadd(111) = T(000) + \Delta_1 + \Delta_2 + \Delta_3$. In contrast, actual mixed-precision graphs can share runtime overhead, experience different memory access patterns, and contain format-conversion effects across layer boundaries. Therefore, accurate deployment policy selection should evaluate complete candidate graphs rather than assuming that isolated single-layer changes sum perfectly.

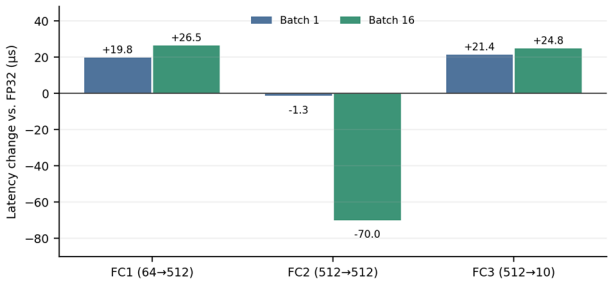


Fig. 5 Effect of quantizing exactly one linear layer at W = 512, measured relative to the all-FP32 network. Negative values indicate faster inference. Values are mean median latency differences across three seeds.

At W = 512 and batch 1, converting FC1 alone adds 19.77 µs, converting FC2 alone saves 1.27 µs, and converting FC3 alone adds 21.45 µs. The additive approximation would therefore predict 85.34 µs for all-INT8, whereas the directly measured all-INT8 result is 60.35 µs. At batch 16, the FC2-only gain is much larger (69.99 µs), while quantizing FC1 or FC3 individually again increases latency. The corresponding additive prediction is 136.49 µs, compared with an observed 87.32 µs. These interaction residuals quantify the inadequacy of the simplistic prediction for the measured backend; they do not by themselves establish which low-level kernel path is responsible.

Table IV. Full-int8 interaction at w = 512

Batch	Additive prediction (µs)	Observed 111 (µs)	Residual (µs)
1	85.34	60.35	-24.99
16	136.49	87.32	-49.18

The differences also help explain the practical behavior of selected masks. For W = 512, batch 1, the 010 configuration uses approximately 411 KiB and averages 44.1 µs, while 111 uses approximately 302 KiB but averages 60.4 µs. The further storage savings of uniform INT8 come with about 37% more request latency compared with the selected mixed configuration. At batch 16, the measured difference between 010 and 111 is small relative to the much larger gap from FP32, so the storage advantage of 111 may be attractive when latency is not the sole constraint. The paper's stated policy uses latency as the primary objective and storage only as a deterministic tie-breaker.

Importantly, single-request latency and batched throughput address different operational questions. A batch-16 invocation amortizes interpreter and operator overhead across multiple images, but its total per-batch latency is larger than one batch-1 invocation. It is therefore inappropriate to interpret a batch-16 model's fast amortized cost as the response time of a single isolated request. The two batch regimes in this study are reported separately for that reason. In a real inference service, the batch distribution, deadline, and available memory would jointly determine which precision mask is deployed.

5.9. Independent Timing Confirmation

Table V reports a second CPU-timing pass with frozen precision masks and different, held-out input tensors. It separately estimates the selected-policy and uniform-INT8 speedup, avoiding reuse of the measurements that optimized the policy. The numerical pattern remains stable: mask 000 is retained at W = 256, batch 1; FC2-only INT8 is effective for the intermediate matrix sizes; and uniform INT8 is selected for W = 1024, batch 16. Most paired confirmation gains are comfortably above one, while the 1.03× gain at W = 512, batch 1 is a small effect relative to virtualized-host timing variability.

Table V. Independent confirmation (mean across three seeds)

W	Batch	FP32 (µs)	Policy (µs)	Policy gain (×)	All-INT8 (×)
256	1	24.2	24.2	1.00±0.00	0.42
256	16	56.6	54.4	1.04±0.02	0.85
512	1	45.3	44.0	1.03±0.01	0.75
512	16	159.6	85.3	1.87±0.11	1.82
1024	1	103.1	63.2	1.63±0.05	1.31
1024	16	675.1	163.9	4.12±0.11	4.12

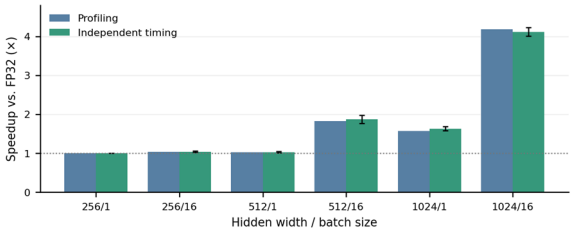


Fig. 6 Profiling versus independent confirmation (three-seed means); error bars denote across-seed standard deviations for confirmation.

The maximum confirmed mean speedup is 4.12× for W = 1024, batch 16 (4.18× in the search pass); for W = 512, batch 16 the corresponding values are 1.87× and 1.83×. For batch 1 at W = 1024, the independently observed gain is 1.63× versus 1.58× during profiling. The selected mask is not slower than either baseline in any of the 18 confirmation comparisons; in three cases it is identically the FP32 model. These results support hardware-specific offline precision assignment while keeping the selection metric and assessment metric distinct. This is a single-CPU confirmation, not an independent-device validation.

6. Conclusion

A measured-latency-driven FP32/INT8 assignment can improve CPU inference by selecting precision per layer at deployment time. The three-layer, eight-mask experiment shows that preferred precision is sensitive to matrix size and

batch shape: small single-request models may remain in FP32, while larger hidden layers benefit from INT8. Across 144 profiling configurations, the policy preserves the paired FP32 test accuracy for every selected mask. Independent retiming with the masks held fixed confirms up to 4.12× mean speedup over FP32 on the measured single-thread x86 backend. The framework provides a lightweight deployment procedure that can be reprofiled for a new backend and expected workload.

References

[1] Ping, W., & Fan, H. (2023). Beyond isolated ratios: Accrual-revenue-cash flow indicators and interaction-aware learning for detecting financial reporting irregularities in U.S. public companies. World Journal of Economics and Business Research, 1(1), 46-55.

[2] Zhang, F., & Yang, J. (2024). Robustness-aware model compression for multimodal perception under challenging conditions. AI and Data Science Journal, 1(1), 103-111.

[3] Fang, Q. (2024). TRUE-EWS: Trustworthy early warning with equalized conformal guarantees for equitable student support in U.S. education. Educational Research and Human Development, 1(2), 44-54.

[4] Jiao, Y., Fan, H., Yue, X., Ping, W., Sun, T., & Wang, J. (2026). Dynamic heterogeneous graph contrastive learning for uncovering collusive financial fraud. Scientific Reports.

[5] Fan, H., Jiao, Y., Wang, M., Chen, J., & Yang, S. (2026). Fraud learns too: Continual graph learning under strategic adversarial drift in dynamic networks. Scientific Reports.

[6] Shen, A., & Liu, W. (2026). Budget-aware adaptive test-time compute with hierarchical self-verification for semantic ID generative recommendation. Computers.

[7] Zhang, H. (2024). Graph-aware multi-task learning for early prediction of timing violations and routing congestion in digital IC physical design. World Journal of Information Technology, 2(1), 13-20.

[8] Zhu, R., & Yue, X. (2024). An explainable machine learning framework for predicting dividend sustainability and financial risk of US REITs under interest-rate shocks. Journal of Trends in Finance and Economics, 1(2), 48-57.

[9] Zhang, C., & Wang, Y. (2024). A data-driven multi-objective production scheduling framework for battery pack manufacturing under material arrival uncertainty. AI and Data Science Journal, 1(1), 86-93.

[10] Zhou, F. (2026). Moment-constrained side-force allocation for wind-assisted merchant ships: Hydrodynamic performance of retractable anti-leeway fins and flap rudders. Innovation and Technology Studies, 3(1), 27-41.

[11] Chen, J., Cui, Y., Zhang, X., Yang, J., & Zhou, M. (2024). Temporal convolutional network for carbon tax projection: A data-driven approach. Applied Sciences, 14(20), 9213. https://doi.org/10.3390/app14209213

[12] Lin, Y., Yue, X., Bian, M., Zhu, R., & Teng, D. (2026). Subgraph-level temporal graph transformer with laundering-motif prototypes for anti-money laundering on blockchains. IEEE Access.

[13] Chen, J., Zhang, X., & Han, X. (2023). Explainable identification of tax-related financial reporting anomalies in U.S. public companies: Cross-fitted tax-consistency residuals and Owen-value Shapley propagation. World Journal of Economics and Business Research, 1(1), 33-45.

[14] Teng, D., Rhee, M., Qin, Y., Zi, B., & Liu, W. (2026). SW-SpeedDLM: Sliding window speculative decoding for diffusion language models under long context constraints. Mathematics, 14(12), 2137. https://doi.org/10.3390/math14122137

[15] Li, P., Feng, E., Wang, G., & Mo, T. (2024). Hybrid deep reinforcement learning and integer optimization for resource scheduling in heterogeneous AI computing clusters. Innovation and Technology Studies, 1(1), 48-56.

[16] Shen, A. (2023). Compute-efficient scaling of large sequential recommendation models under dynamic user behavior. World Journal of Information Technology, 1(1), 62-71.

[17] Yue, X., & Zhu, R. (2024). An explainable hybrid machine learning framework for cash-flow-conditioned multi-horizon liquidity risk early warning: An SME-oriented benchmark study. World Journal of Management Science, 2(1), 63-72.

[18] Cao, J., Zheng, W., Ge, Y., & Wang, J. (2025). DriftShield: Autonomous fraud detection via actor-critic reinforcement learning with dynamic feature reweighting. IEEE Open Journal of the Computer Society, 6, 1166-1177.

[19] Guo, Z., Chen, T., & Shang, W. (2022). Class-conditional intermediate-domain adversarial adaptation for cross-device acoustic scene classification. Journal of Computer Science and Electrical Engineering, 4(1), 26-35.

[20] Zhao, X., Liu, J., Wang, Y., & Wang, J. (2026). CryptoMamba-SSM: Linear complexity state space models for cryptocurrency volatility prediction. IEEE Open Journal of the Computer Society, 7, 226-243.

[21] Mai, N. T., Fang, Q., & Cao, W. (2025). Measuring student trust and over-reliance on AI tutors: Implications for STEM learning outcomes. International Journal of Social Sciences and English Literature, 9(12), 11-17.

[22] Teng, D. (2025). RSRBench: Reproducibility and performance benchmarking for regional seismic risk workflows. Academic Journal of Architecture and Civil Engineering, 3(1), 40-52.

[23] Zhang, F., Shen, A., & Huang, S. (2026). TRIAD: Co-designing mixed-precision quantization, operator fusion, and on-chip memory reuse for real-time vision-language-action inference on automotive NPUs. Results in Engineering, 113673.

[24] Fan, H., Liang, Y., Jiao, Y., Ping, W., & Lin, Y. (2026). An explainable temporal machine learning framework for financial fraud risk prediction in listed institutions. Highlights in Business, Economics and Management, 70, 173-185.

[25] Wang, J., Liu, J., Zheng, W., & Ge, Y. (2025). Temporal heterogeneous graph contrastive learning for fraud detection in credit card transactions. IEEE Access.

[26] Wang, Y., Bian, M., & Lin, Y. (2022). An explainable temporal graph neural network for disruption prediction and risk propagation in multi-tier supply chains. Journal of Computer Science and Electrical Engineering, 4(1), 17-25.

[27] Chen, T., Shang, W., & Guo, Z. (2024). XSLA: An SLA-aware heterogeneous XPU scheduling framework for edge AI inference. World Journal of Information Technology, 2(2), 48-59.

[28] Liang, Y., & Jiao, Y. (2024). Real-time financial fraud detection using a feature-light temporal dynamic graph neural network with contrastive learning. Journal of Trends in Finance and Economics, 1(1), 76-84.

[29] Shang, W., & Chen, T. (2024). PrefixKD-Edge: Real-time locomotion-intent prediction with prefix self-distillation on lightweight temporal networks. Innovation and Technology Studies, 1(1), 30-38.

[30] Guo, Z., Fang, Q., Yang, J. S., & Rhee, M. (2026). Closed-loop audio neuroregulation for ADHD: Graph-based attention decoding and rhythm entrainment. IEEE Access.

[31] Mo, T., Wang, G., Feng, E., & Li, P. (2023). Execution-feedback-guided context selection for large language model-

based software testing and debugging. World Journal of Information and Knowledge Management, 1(1), 55-64.

[32] Jiao, Y., & Liang, Y. (2024). An integrated process-mining and explainable machine-learning framework for bottleneck and internal-control risk detection in the multi-entity financial close. Social Science and Management, 1(3), 96-107.

[33] Lin, Y., Wang, Y., & Bian, M. (2024). XOR-EW: An explainable and cost-aware artificial intelligence framework for operational risk early warning in financial institutions. AI and Data Science Journal, 1(1), 94-102.

[34] Qin, Y., Rhee, M., Teng, D., Zhang, C., & Zi, B. (2026). Neuro-symbolic constraint verification for LLM-driven internet finance transaction execution. Scientific Reports.

[35] Wang, J., Tan, Y., Jiang, B., Wu, B., & Liu, W. (2025). Dynamic marketing uplift modeling: A symmetry-preserving framework integrating causal forests with deep reinforcement learning for personalized intervention strategies. Symmetry, 17(4), 610. https://doi.org/10.3390/sym17040610

[36] Fang, Q., & Liu, W. (2025). HARLA-ED: Resolving information asymmetry and enhancing algorithmic symmetry in intelligent educational assessment via hybrid reinforcement learning. Symmetry, 18(1), 58. https://doi.org/10.3390/sym18010058

[37] Qin, Y., Zou, J., & Rhee, M. (2022). An AI-driven intelligent transaction execution framework for internet finance platforms: Cost-sensitive real-time integrity interdiction. Eurasia Journal of Science and Technology, 4(2), 36-44.

[38] Rhee, M., Zou, J., & Qin, Y. (2024). CLARITY: A closed loop framework for duplicate aware failure prediction and budgeted automated remediation in cloud infrastructure. AI and Data Science Journal, 1(1), 76-85.

[39] Bian, M., Wang, Y., & Lin, Y. (2024). Explainable supplier risk prediction for critical telecommunications infrastructure procurement: An empirical XGBoost-SHAP framework. Social Science and Management, 1(1), 64-77.

[40] Wang, G., Li, P., Feng, E., Shen, A., & Rhee, M. (2026). A dual-process memory-augmented large language model agent for explainable sequential recommendation. IEEE Access.

[41] Feng, E., Li, P., Mo, T., & Wang, G. (2022). PRISM: Prediction-guided risk-aware deep reinforcement learning for energy-aware dynamic resource scheduling in distributed cloud data centers. Journal of Computer Science and Electrical Engineering, 4(1), 53-61.

[42] Chen, J., Han, X., Zhang, X., & Yang, Y. (2023). Exact concept-level explanations for early detection of financial misstatements. Eurasia Journal of Science and Technology, 5(2), 34-48.

[43] Yue, X., Yang, J., & Liu, W. (2026). FraudDebate-Agent: A multi-agent LLM framework with an evidence-based debate mechanism for financial statement fraud detection. Mathematics, 14(15), 2695. https://doi.org/10.3390/math14152695

[44] Mo, T., Zhang, C., Zou, J., Guo, Z., & Rhee, M. (2026). Self-evolving AI agents with dual memory for automated software testing and bug localization. IEEE Access.

[45] Zhang, F., & Yang, J. (2022). Accuracy-aware model compression for real-time visual perception under resource constraints. Journal of Computer Science and Electrical Engineering, 4(1), 45-52.

[46] Liang, Y., Jiao, Y., Ping, W., Fan, H., & Han, X. (2026). Adaptive event-driven labeling: A neuro-symbolic multiagent framework for causal inference in non-stationary time series. IEEE Access.

[47] Wang, G., Li, P., Feng, E., & Mo, T. (2024). Uncertainty-aware LLM-as-a-judge evaluation for reliable search and recommendation systems: Calibrated confidence and impact-aware selective human review. AI and Data Science Journal, 1(1), 112-121.

[48] Chen, J., Liu, J., Liang, Y., & Zhou, M. (2026). HeteroGCL: A heterogeneous graph contrastive learning framework for scalable and sustainable cryptocurrency AML. Applied Sciences, 16(6), 2860. https://doi.org/10.3390/app16062860

[49] Shen, A., & Huang, S. (2022). A systems-aware framework for intelligent large-scale recommendation models. Journal of Computer Science and Electrical Engineering, 4(1), 36-44.

[50] Chen, J., Liang, Y., Liu, J., & Zhou, M. (2026). Temporal transformer with conditional tabular GAN for credit card fraud detection: A sequential deep learning approach. Mathematics, 14(7), 1183. https://doi.org/10.3390/math14071183

[51] Mai, N. T., Cao, W., & Fang, Q. (2025). A study on how LLMs (e.g., GPT-4, chatbots) are being integrated to support tutoring, essay feedback and content generation. Journal of Computing and Electronic Information Management, 18(3), 43-52.

[52] Fang, Q. (2024). Bias-aware large language models for equitable educational feedback and assessment. Trends in Social Sciences and Humanities Research, 2(1), 41-50.

[53] Wang, J., Ge, Y., Liu, J., & Jiang, B. (2024). Early warning of liquidity stress in U.S. banks with seasonally aligned frailty hazards and delayed-feedback conformal alarms. Journal of Trends in Finance and Economics, 1(1), 85-100.

[54] Zou, J., Qin, Y., & Rhee, M. (2024). Beyond model scale: A task-aware reliability benchmark for large language models in clinical decision support. Innovation and Technology Studies, 1(1), 39-47.

[55] Ping, W., & Fan, H. (2024). Disclosure quality and valuation risk in level 3 investments: A disclosure-implied risk framework for U.S. business development companies. Social Science and Management, 1(1), 78-87.

[56] Wang, J., Liu, J., Ge, Y., & Jiang, B. (2023). Risk-aware ensemble learning for credit default prediction under economic uncertainty. World Journal of Economics and Business Research, 1(1), 18-32.

[57] Teng, D. (2025). TRIAD: Deadline- and reliability-aware scheduling for post-earthquake structural assessment workflows. World Journal of Engineering Research, 3(6), 55-66.

[58] Wang, Y., & Zhang, C. (2023). AERIS: Adaptive executor-scoped resource management for privacy-preserving data processing in Apache Spark. Eurasia Journal of Science and Technology, 5(2), 25-33.

[59] Sun, T., Yang, J., Li, J., Chen, J., Liu, M., Fan, L., & Wang, X. (2024). Enhancing auto insurance risk evaluation with transformer and SHAP. IEEE Access, 12, 116546-116557.

[60] Wang, B., Zhao, W., & Wang, Z. (2024). DAS-GNN: A scalable disagreement-aware graph neural framework for anomaly detection in large-scale networks. AI and Data Science Journal, 1(1), 67-75.