YOLOv4 on Tenstorrent Wormhole

An implementation-level tour of the m640 configuration: how the host, model graph, Tensix kernels and post-processing fit together, what changed, and what the measurements actually show.

SEMICON Europa 2026 | Tenstorrent track | YOLOv4 | Wormhole n150 | Selected config: m640

1. Result at a glance

17.8 → 2.106 ms
Practice-set wall time per source image; fastest-of-repeats measurements

8.45×
Throughput speed-up over the organisers' fast reference, using 17.8 ms as the practice baseline

.464 → .5586
Practice mAP@50; +.0946 absolute, or +9.46 mAP points

4 → 1
320 × 320 source images packed into one 640 × 640 model pass

Measurement	Reference	m640, custom kernels + box-step v2	Change
Practice set (200 images)	17.8 ms / .464 mAP	2.106 ms / .5586 mAP	**8.45× faster; +.0946 mAP**
COCO1k comparison	m640 base: 2.416 ms / .6392	m640 kernels + v2: 1.672 ms / .6439	**30.8% less time; +.0047 mAP**

> **Read the numbers correctly.** Practice and COCO1k are different evaluation sets, so their mAP values should not be compared to each other. Practice is used for the headline against the supplied reference; COCO1k is a larger engineering comparison. The official 800-image scoring run is separate; stress-test draws are not a substitute for that result.

The 2.106 ms is **wall-clock throughput per original photo**, computed from the evaluator's total run time divided by the number of input photos. It is not the latency of processing one photo alone: m640 processes four photos together and shares card-pass overhead between them.

2. The whole journey: from baseline to m640

This project did not start with custom kernels. We first measured the end-to-end submission, identified where time and accuracy were being lost, fixed the highest-impact causes, and only then built device kernels and chose the operating point. Each phase changed what the next phase needed to optimize.

Phase	What we found or changed	Measured milestone	Why that led to the next step
0. Establish a real baseline	The unoptimized submission launched a large graph of individual operations and spent substantial time in the host path. We recorded both the plain reference and the organisers' traced fast reference.	Plain reference: 85.3 ms/photo; fast reference: 17.8 ms/photo and .464 practice mAP@50.	Tracing removed much of the Python launch overhead, but the remaining 17.8 ms still included avoidable input, output and graph work.
1. Keep the card fed	Moved decode/resize and tensor preparation into parallel workers, limited nested PyTorch threading, and overlapped output processing with the next card pass.	Input-path stage: 17.6 → 12.9 ms/frame; input plus overlapped output reached 5.9 ms/frame in an early pipeline measurement.	Those changes exposed the next bottlenecks: box decoding and many small card operations. These are stage measurements, not directly comparable final m640 benchmarks.
2. Recover accuracy before chasing more speed	The card was .094 below the fp32 CPU reference. Higher HiFi modes did not fix it. Inspection found HardMish substituted for Mish in 72 locations; replacing the approximation with real Mish restored the activation the weights were trained for.	At 320, library Mish raised practice mAP from .464 to .5145, costing about 0.7 ms. Real Mish, fp32 accumulation and multi-label NMS brought a later 320 card pipeline to .5662, matching the .5667 fp32 CPU ceiling with multi-label.	Accuracy was no longer the reason to avoid the card. The extra cost of faithful Mish became a prime target for a custom SFPU implementation.
3. Move box generation out of the card's small-op tail	Replaced slow framework-heavy NMS operations with a faster exact path; enabled multi-label detections; then stopped the card after raw detector heads and decoded them on the host using compiled code.	Raw-head change: 5.95 → 3.74 ms/frame in the earlier pipeline. The exact box step dropped from 7.30 to 4.05 ms on one core (identical detections), and to 3.35 ms in the later offline benchmark; torchvision batched NMS then reached 1.38 ms at the same mAP. Multi-label added about +.022 mAP at the same speed.	Even after host and output improvements, the model graph had a high fixed cost per pass. The next large lever was to process more source photos per pass.

Phase	What we found or changed	Measured milestone	Why that led to the next step
4. Expand the speed/accuracy frontier—and debug 640	Built resolution variants and tested mosaics. The first 640 path produced bad features because a neck convolution read a mismatched shard layout: the 320-only branch explicitly resharded its input, while the larger-resolution branch skipped that step. We enabled resharding above 320 (`reshard_if_not_optimal = resolution > 320`) and rechecked the stages. Mo's hardware report also notes that 16-bit accumulation degraded 640 output (PCC 0.768) and that fp32 accumulation fixed it.	The pre-fix 4 × 320 mosaic measured about 3.97 ms/photo but only .4447 mAP. After fixing the layout and searching 28 graph/precision settings, the no-kernel m640 point reached 2.755 ms and .5581 practice mAP (2.389 ms / .6392 on COCO1k in that matrix).	The layout issue showed that a fast result is meaningless if tensor placement is wrong. After the fix, m640 became a valid accuracy/speed knee and the target for kernel work.
5. Replace the remaining costly device patterns	Implemented four kernels: a faster Mish SFPU path; Mish plus residual addition; an SPP max-pool kernel; and a sharding-aware upsample/concat kernel. Also removed unnecessary moves/conversions and overlapped output transfer using a second command queue.	In the 320 kernel-stack comparison, 3.924 → 3.342 ms/frame in the same job. After porting to all six configurations, the stack reduced COCO1k time by about 24–30%, depending on the point.	We generalized kernels only after establishing the Pareto configurations. That avoided optimizing a single input size that the final system would not use.
6. Improve ranking and validate the combination	Added box-step v2: object/class score re-ranking, fused box voting and a mosaic tile-edge penalty. Then measured it together with the custom kernels rather than assuming independent gains would stack.	Across six COCO1k configurations, v2 improved mAP by +.0037 on average (held-out half: +.0034 average, positive on all six). The m640 kernel+v2 cross-check measured 1.672 ms / .6439 mAP on COCO1k.	The combination check established the submitted stack and gave us evidence for both runtime and accuracy without relying on separate ablations.
7. Select m640 for the scored run	Compared the fastest mosaic against the larger-margin m640 point, the single-image resolutions, and the scoring rules.	Selected m640 with kernels + v2: 2.106 ms/photo and .5586 practice mAP. Against 17.8 ms / .464, that is about 8.45× faster and +.0946 mAP.	m512 is faster but has a narrower accuracy margin. m640 sacrifices some speed to keep a much larger buffer above the score's .447 accuracy floor.

> **Do not sum the phase timings.** The 12.9, 5.9, 3.74 and 3.35 ms figures come from different stages and test setups; some optimizations overlap. The defensible end-to-end comparison is the same-set reference versus the selected m640 run, with intermediate numbers used to explain how the bottlenecks changed.

The rest of this document unpacks the final m640 architecture and explains the mechanisms behind each phase.

3. End-to-end architecture

The model remains YOLOv4: the work was in restoring its intended behavior, selecting an efficient input/execution mode, reducing framework overhead, and replacing expensive operations with targeted kernels. The weights and YOLO architecture were not replaced with a different detector.

1. Host input
Read and resize images to 320 × 320; convert to contiguous tensors and prepare uploads.

→

2. m640 mosaic
Place four images in a 2 × 2 layout in a 640 × 640 canvas.

→

3. YOLOv4 on card
Backbone → SPP → PANet neck → three detection heads; custom kernels replace selected costly ops.

→

4. Raw heads to host
Skip the card-side genboxes tail; transfer compact head tensors for host decode.

→

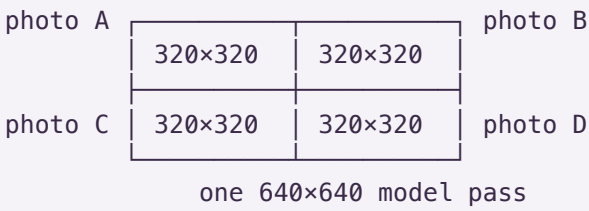
5. Per-image detections
Decode, assign mosaic predictions to their source tile, rescale boxes, multi-label NMS, return results.

Where the model's pieces sit

YOLOv4 stage	Purpose	Relevant optimization
Backbone (CSPDarknet53)	Build increasingly rich visual features as spatial size shrinks and channel depth grows.	Correct Mish activation; K1 custom Mish; K2 fused Mish plus residual add.
SPP	Combine local and wider context with max-pool windows 5, 9 and 13.	K3 replaces a sequence of small graph operations with one multi-core kernel.
PANet neck	Upsample and combine coarse and fine feature maps.	K4 writes upsampled/concatenated data in the next convolution's sharded layout.
Three detection heads	Predict boxes at strides 8, 16 and 32 for different object scales.	Return raw head outputs to the host instead of running genboxes as many small card ops.

4. Why the selected configuration is m640

m640 is a **four-image mosaic using 320-pixel tiles**. It is not four native 640 × 640 inferences and does not give each photo 640-pixel detail. Each source photo is resized to 320 × 320, then placed in one quadrant of a 640 × 640 canvas.



The card's graph has a mostly fixed cost per pass: launching and scheduling hundreds of operations costs time even when each operation does little arithmetic. Four photos in one pass amortize much of that fixed cost. The model still processes the pixel work for all four tiles, but the per-pass overhead is shared.

Correctly recovering each photo's boxes

1. Resize each input photo to its 320 × 320 tile and place it in the 2 × 2 canvas.
2. Run the 640 × 640 YOLO graph once.
3. For each predicted box, use the grid cell that produced it to identify its tile. The implementation groups that tile's cells back into the normal per-image head layout.
4. Decode and rescale each tile's boxes in its own 320 × 320 coordinate system, then map them to the original photo dimensions.
5. Apply a small confidence penalty to boxes near internal tile edges, where an object may be cut by the mosaic boundary.

Because each prediction belongs to the cell that emitted it, tile ownership can be recovered from the head-grid location rather than guessing from the final box center. This matters when predicted boxes extend beyond a tile boundary.

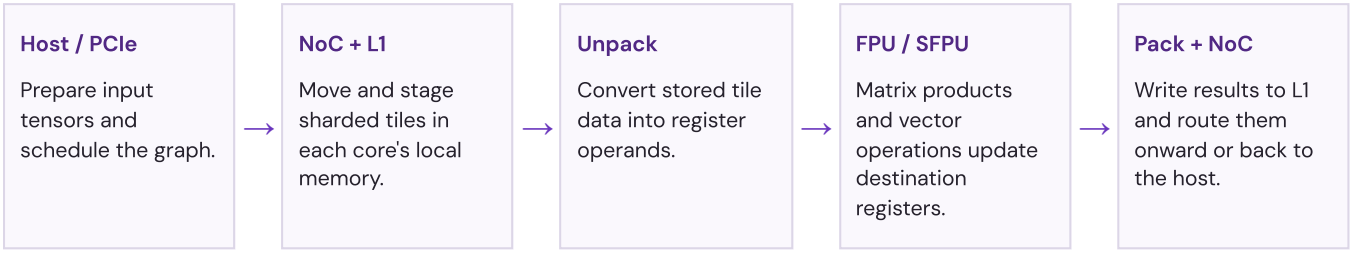
Why this point on the speed/accuracy curve?

Config, kernels + v2	Practice ms/photo	Practice mAP@50	Interpretation
m512 (4 × 256)	1.640	.4787	Fastest, but only .0317 above the .447 accuracy floor.
m640 (4 × 320)	**2.106**	**.5586**	Chosen knee: still very fast, with a much larger accuracy margin.
r512 (single image)	4.597	.6987	Higher accuracy, but more than twice m640's practice time.

m640's practice mAP is .1116 above the scoring floor of .447. The choice gives up some accuracy versus larger single-image inputs, but keeps substantial headroom while retaining a large speed advantage.

5. How the accelerator runs it

The Wormhole n150 provides an 8 × 8 grid of 64 Tensix cores. Each core has local SRAM (L1), data-movement engines, and separate matrix and vector math units. The host CPU prepares work and orchestrates the device; it does not replace the card's neural-network computation.



- **FPU:** the matrix engine used for convolution/matmul work and additions.
- **SFPU:** the vector unit used for elementwise functions such as Mish and max operations.
- **Unpacker and packer:** move between L1 tile formats and registers. Tile data is distributed across cores, so layout and sharding determine whether an operation can avoid copies.
- **Command queues and traces:** record/replay the model work and overlap transfers with computation where dependencies permit.

> **Important K2 detail:** its SFPU Mish and FPU residual addition are issued back-to-back in the compute instruction stream. We fuse the dataflow and reuse the destination registers; we do not claim that SFPU and FPU execute the same element simultaneously.

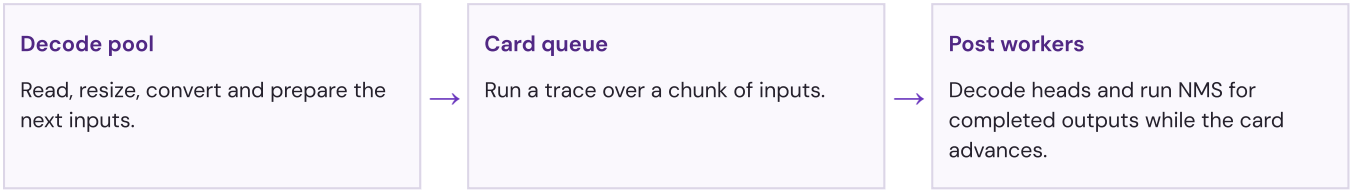
Precision in the selected m640 graph

Part	Representation / mode	Reason
Weights	bfp8 block floating point	Compact storage; groups of 16 values share an exponent.
Activations in the 640 graph	bf16	Matches the graph's supported tensor and kernel variants.
Most matrix operations	LoFi mode	Best measured speed/accuracy choice in the tested matrix; raising fidelity alone did not recover the main accuracy loss.
First convolution (`PREC_FIRST=1`)	bf16 input/weights, HiFi4 and fp32 accumulation	A small, targeted precision increase that was beneficial or neutral in nearly all measured pairs.
Selected m640 graph accumulation (`ACC=0`)	Do not enable fp32 accumulation for every convolution	The matrix showed that blanket accumulation was not a useful accuracy/speed trade-off for this mosaic point.

The custom kernels compute in the destination registers and the packer writes the output in the tensor's format (bf16 for m640).

6. Host pipeline: keep work moving

The reference path did not spend all its time doing neural-network arithmetic. Decode, conversion, Python dispatch and post-processing could leave the card waiting. The changes overlapped independent stages and reduced CPU scheduling overhead.



Change	Mechanism	Evidence / scope	Trade-off
Parallel input preparation	Eight decoder workers perform OpenCV decode/resize and prepare contiguous bf16 tensors; conversion/upload work moves out of the serial main-thread path.	Input-path stage: 17.6 → 12.9 ms/frame in its early isolated measurement.	Uses host CPU cores and memory bandwidth; requires controlling nested library thread pools.
Limit PyTorch threads	`TORCH_THREADS=1` avoids each worker creating a large competing thread team.	Removed oversubscription and improved end-to-end throughput.	Optimal in this multi-worker pipeline; not a universal setting for every workload.
Overlap output work	Two post workers handle completed output while the card runs the next chunk.	With the input-path work, early pipeline measurement reached 5.9 ms/frame.	Only hides post-processing while it remains shorter than the card stage.
Numba nogil box step and GC control	Compile the hot decode/NMS path and avoid Python GIL convoying; freeze cyclic GC during the timed run.	Reduced host-side contention and timing spread; results remain determined by the same model outputs.	More implementation complexity; garbage collection is restored outside the timed interval.
Fill/drain schedule (I5)	Start and finish with smaller chunks (8, 16, …, 16, 8) so the card need not wait for a full chunk at either edge of a run.	Saved 26 ms over 1,000 COCO1k images; the fixed saving amortizes over the evaluation set.	Changes only pipeline boundary overhead; it is not a per-image model speed-up of 26 ms.

7. Accuracy recovery: restore the model's intended activation

The card port initially scored .464 mAP@50, while the same weights in an fp32 CPU reference scored .558. Increasing precision did not fix the gap: bf16 weights with HiFi4 scored .445, no better than the .464 LoFi baseline. The investigation found that the port had substituted HardMish for the YOLOv4 model's Mish activation in 72 locations.

Function	Definition	Behavior that matters
Mish	`x · tanh(ln(1 + e^x))`	Smooth; retains small negative outputs for negative inputs.
HardMish	`x · clamp(x + 2, 0, 2) / 2`	Cheaper piecewise approximation; differs most in common activation ranges and cuts off negative tails.

Replacing Mish with HardMish in the fp32 reference reproduced most of the observed accuracy loss (.558 → .479, versus .464 on the card). Restoring real Mish on the card improved the 320 practice result from .464 to .5145 (about +.05 mAP), at roughly +0.7 ms for the library implementation. That accuracy fix motivated the custom Mish kernel, which made the more faithful activation much cheaper.

Precision was treated as a measured knob, not a blanket fix

- Fp32 accumulation helped at 320 (+.024 mAP for about +0.22 ms) but did not consistently help the larger or mosaic graphs; the selected m640 uses `ACC=0`.
- The first convolution is a targeted exception: `PREC_FIRST=1` applies bf16 with HiFi4 and fp32 accumulation. It is a small portion of the graph and measured as beneficial or neutral in 13 of 14 comparisons.
- Using bf16 activations everywhere at 320 crashed the graph. Precision changes therefore remained constrained by the graph's supported layouts and types.

8. Detection heads and box processing

Return raw heads instead of hundreds of tiny card operations

The standard model's genboxes tail performs sigmoid/exp, reshapes and concatenations to turn the three detector heads into candidate boxes. It was implemented as many small operations with meaningful dispatch overhead. The raw-head path stops after the head convolutions, packs the outputs, and transfers them to the host. The host decodes candidates in compiled code and applies the same coordinate transforms and filtering logic.

At 320, the genboxes tail was about 229 operations and 1.133 ms of a 3.858 ms frame. Moving this tail off-card reduced a measured pipeline from 5.95 to 3.74 ms/frame. Those figures are from the earlier 320 pipeline, not an isolated m640 ablation.

Faster exact post-processing, then an accuracy-oriented variant

Piece	What changed	Effect / caveat
Fast exact NMS path	Use an integer trick for class maxima, process present classes, compute IoU in fp16 matrices with the reference rounding, and reduce framework-call overhead.	Bit-identical detections for the exact path; offline box step 7.30 → 4.05 ms (one core), 3.35 ms in the later benchmark; torchvision batched NMS 1.38 ms at the same .4640 mAP.
Multi-label NMS	Keep a candidate for every class above threshold instead of assigning each box only its highest-scoring class.	Recovered recall; measured +.022 mAP at the same speed.
m640 settings	Multi-label, NMS IoU 0.45, confidence floor 0.001, up to 100 detections per image/class.	Selected from offline box-step sweeps; low-score candidates may improve recall under the evaluator's AP ranking.
Box-step v2	Rank using `sigmoid(obj)^0.5 · sigmoid(cls)^1.25`; fused box voting for overlapping boxes (IoU > 0.75); apply a 0.5 score factor near mosaic tile edges.	Host-only; m640 practice .5581 → .5597 before the kernel cross-check, with no measurable added latency. Combined m640 kernels + v2 practice result: .5586.

The mAP metric is AP at IoU 0.5, averaged over classes with ground truth. The post-processing changes deliberately emphasize recall and ranking. Extra candidates cost host work, so the confidence floor and NMS settings were measured rather than lowered without limit.

9. Four custom Tensix kernels

The kernels target different sources of cost: one expensive nonlinear function, repeated graph boundaries, a multi-op pooling block, and layout conversions in the neck. They were implemented as custom device operations and ported to the six selected configurations, including bf16 variants for the larger graphs.

Kernel	Original work	Implementation	Measured effect and precision notes
K1: Mish	Repeated elementwise Mish calls after convolutions.	Compute an algebraically equivalent Mish form, `x − 2x / ((1 + e^x)^2 + 1)`. Approximate the exponential and reciprocal with hand-scheduled SFPU operations; interleave two rows to keep the instruction stream occupied. Separate kernels handle bfp8 and bf16 paths.	One 25,600 × 64 Mish call: library 54.6 µs, V12 27.7 µs, V14 15.5 µs (~3.5×). The m640 graph is bf16, so it runs V34 (V14 plus round-to-nearest bf16): on a 62-core bf16 shard, 105.0 → 30.7 µs, max abs error .0101 vs fp32 Mish, PCC ≥ 0.99992. These are per-call benchmarks, not a 3.5× whole-model speed-up.
K2: Mish + residual add	At 23 ResBlock tails, compute Mish, store/reload, then add the skip tensor.	Load the operands into the on-core dataflow, run Mish on the SFPU, then issue the FPU add using the destination registers; write the final result once.	Removes 23 graph ops/frame and an intermediate write/read. At bfp8, it also avoids an intermediate Mish rounding before the add; m640's activations are bf16, so the key m640 win is the fused op/dataflow rather than claiming that bfp8-specific rounding benefit.
K3: SPP max-pool	Three spatial max-pool windows (5, 9, 13) and their intermediate graph operations.	Use the identity that a larger max window can be built from cascaded smaller max windows. A multi-core operation gathers sharded data and computes the required max values in a controlled layout.	In the isolated 320 benchmark, 136.8 → 16.7 µs (8.2×); the SPP output is bit-exact. The general kernel replaces a 12-op block with one operation.
K4: upsample + concat	Two neck paths with upsample, layout changes and concatenation before the following convolution.	Use the NoC/data-movement cores to gather source tiles, repeat rows for 2× upsampling and write directly into the next convolution's sharded layout.	About nine operations become one per chain; byte-exact. Isolated traced saving was about 83 µs across the two chains.

Combined kernel result

On m640, the custom stack reduced COCO1k time from 2.416 ms/photo in the base folder to 1.672 ms/photo with kernels plus box-step v2. The comparison combines several device kernels and the host-only v2 change, so it is not a clean attribution of all 30.8% to one kernel. The paired COCO1k result was .6392 → .6439 mAP@50; the measured difference is small relative to dataset variation, not evidence that the kernels alone improve accuracy.

10. Runtime and I/O changes on the card

Remove work that does not change the tensor values

- Disable a halo-output reallocation that copied convolution outputs into fresh buffers; 20 Move operations disappeared in the tested graph.
- Skip an unnecessary input Move.
- Skip three `add(x, 0)` operations that only converted dtype where the following operation accepted the tensor directly.
- Copy output to a DRAM buffer and read it on command queue 1 with events, allowing the download for one pass to overlap the next trace where dependencies allow.

The combined config/I/O patch reduced the measured graph from 401 to 379 operations, cut the inter-frame idle gap from 114 to 4 µs, and saved roughly 0.12–0.23 ms/frame in the original ablation. The exact gain varies with graph and batch mode; the operation count is not a claim that every removed operation saves the same time.

Why reducing operation count matters

The 320 graph was estimated at roughly 10 GMAC, below a millisecond of idealized arithmetic on the card. In practice, many small graph operations and dispatch gaps dominated. Therefore the design prioritized fewer launches, fused dataflow, sharding-aware writes and overlapping transfers over simply increasing arithmetic precision or peak compute.

11. What drove the end-to-end improvement

The following measurements come from different stages of development. They show the direction and scale of each lever, but they are **not additive**: some measurements use different input sizes, datasets, timing protocols or overlapping optimizations.

Stage / lever	Measured result	How to interpret it
Organisers' fast reference	17.8 ms/photo, .464 practice mAP	Starting headline on the supplied 200-image practice set.
Input preparation alone	17.6 → 12.9 ms/frame	Early isolated input-pipeline measurement; not the final m640 comparison.
Input preparation + overlapped output	5.9 ms/frame in the early pipeline	Decode, card and post-processing overlap; later changes further reduce time.
Raw heads + host decode	5.95 → 3.74 ms/frame in the earlier pipeline	Moves genboxes from many small card ops to the host.
m640 mosaic, base kernels off / box-step v2 off	2.755 ms/photo, .5581 practice mAP	Four 320 inputs share one 640 model pass, with the selected graph and host pipeline.
Selected m640, kernels + v2	**2.106 ms/photo, .5586 practice mAP**	**Final selected setting measured on practice: about 8.45× reference throughput and +.0946 absolute mAP.**

Cross-checks and repeatability

- The kernels and box-step v2 were run together on all six configurations; m640 practice and COCO1k measurements are in that cross-check.
- The custom kernel stack had zero fallbacks in the six-folder runs; unsupported shapes would otherwise use stock operations.
- A broader stress suite reported zero crashes or missing outputs across 396 runs and included malformed/edge files. Its synthetic hidden-like 800-image draws are a robustness check, not the official hidden-set score.
- Accuracy comparisons use paired evaluation where possible; COCO1k bootstrap intervals are more informative than small practice-set differences, especially for rare classes.

12. Trade-offs and approaches we rejected

Choice or experiment	Benefit	Cost, failure mode or reason to reject
m640 mosaic rather than m512 mosaic	Much larger practice accuracy margin while remaining far faster than the reference.	About 0.47 ms/photo slower in the combined practice result; each photo still has 320-pixel detail.
Four-image mosaic rather than native high-resolution images	Amortizes graph and dispatch overhead across four inputs.	Objects are represented at tile resolution; objects crossing mosaic seams may be harder to detect. Edge penalty partly addresses this.
Restore Mish, then optimize it	Recovers model behavior expected by the trained weights.	Library Mish costs extra time; custom approximation must be checked for numeric error.
LoFi for most matrix work	Fast and the best measured general setting.	Does not eliminate all quantization error; raising fidelity alone did not help (320: HiFi4 without fp32 accumulation .5099, with it .5357).
First-layer precision exception	Targeted accuracy improvement at negligible measured runtime cost.	Requires special configuration and still follows the graph's dtype constraints.
Letterbox resize	Preserves aspect ratio.	Reduced mAP by about 7 points because the model was trained using stretched images.
Flip test-time augmentation	Small accuracy improvement.	About twice the card work for roughly +0.4 mAP points; poor score trade-off.
FP32 accumulation in every convolution	Helped one 320 operating point.	Did not consistently improve larger/mosaic settings and adds time; m640 uses the measured ACC=0 setting.
640 graph before layout fix	Would enable a larger graph and mosaic.	Incorrect layout at neck conv9 caused corrupted features. Explicitly resharding the input for resolutions above 320 fixed the issue.
Alternative NMS variants and on-card argmax	Potentially reduce candidates or alter suppression.	Tested variants were slower or lower-mAP; on-card argmax measured 5.4–12.6 ms.
Fuse Mish into the stock convolution kernel	Would remove a separate activation operation.	Tested stock-fused path was about 0.25 ms slower overall; a kernel overlay passed unit tests but hung on the full model.

Scoring strategy: m640 is a robust speed/accuracy point rather than the absolute fastest point. It is roughly 8× faster than the reference and well above the .447 accuracy floor on practice. That provides more safety against hidden-set variation than m512's narrower margin.

13. Team contributions and ownership

The final pipeline is a team integration. The documented work spans host input, model numerics, post-processing, custom device kernels and evaluation:

Workstream	Contributions
Nick	Output-stage optimization and exact fast box processing; HardMish accuracy diagnosis; multi-label evaluation; custom SFPU Mish, fused Mish/residual, SPP and upsample/concat kernels; config/I/O patch; kernel integration across the six configurations; paired-bootstrap tooling and technical documentation.
Chanyoung	Input pipeline, raw-head route, Numba box path and GC control, fill/drain scheduling, mosaic/raw-head integration, configuration matrix, frozen combo folders, box-step v2 and stress testing.
Mo	Real-Mish and precision experiments, resolution variants, the 640 layout diagnosis/fix, and evaluation sets.
Other team work	Additional model and integration work completed by the fourth team member; the implementation details are not itemized here.

This attribution reflects the project notes. End-to-end latency and mAP figures are team results, not single-person results.

14. Short glossary

Term	Meaning in this project
m640	One 640 × 640 model canvas containing four 320 × 320 source-image tiles.
mAP@50	Mean average precision at intersection-over-union threshold 0.5.
FPU	Matrix-oriented math engine in each Tensix core.
SFPU	Vector-oriented math engine used for elementwise functions and reductions in these kernels.
NoC	Network-on-chip interconnect used to move tiles between cores and memory.
L1	Per-core local SRAM holding tiles and circular-buffer data.
bfp8	Block floating point format: a shared exponent for groups of values, reducing storage.
bf16	16-bit brain floating point format, used for activations in the larger graph.
Raw heads	Un-decoded outputs of the three YOLO detection heads, processed into boxes on the host.
Paired bootstrap	Resample the same evaluation images for two variants to estimate uncertainty in their mAP difference.

15. The complete YOLOv4 configuration set

The team did not tune only m640. We measured six points on the speed/accuracy frontier, then applied the custom kernels and box-step v2 across all six. The table below puts the selected setting in context and closes the journey with the alternatives it was chosen against.

Configuration	Input per card pass	Practice ms/photo	Practice mAP@50	COCO1k ms/photo	COCO1k mAP@50	Trade-off
m512_a0p1	4 × 256 mosaic	1.640	.4787	1.216	.5697	Fastest point; smallest margin above the practice accuracy floor.
m640_a0p1	**4 × 320 mosaic**	**2.106**	**.5586**	**1.672**	**.6439**	**Selected: strong speed with a substantially safer accuracy margin.**
r320_a1p1	1 × 320	3.230	.5588	2.999	.6453	Similar practice accuracy to m640, but slower per source photo.
r384_a0p0	1 × 384	3.815	.6369	3.568	.6934	More detail and accuracy, at a higher throughput cost.
r512_a0p1	1 × 512	4.597	.6987	4.340	.7522	High accuracy; slower than the m640 scoring target.
r640_a1p1	1 × 640	7.150	.7330	6.810	.7524	Highest practice accuracy; COCO1k is nearly tied with r512 for much more time.

All rows are the measured kernels + box-step-v2 cross-check. Practice has 200 images; COCO1k has 1,000. Timings are per source image, including batching/mosaic throughput. The suffixes encode the chosen accumulation (`a0/a1`) and first-layer precision (`p0/p1`) settings; mosaic rows process four source images together.

Why the team selected m640

- **m512** wins on raw throughput, but its practice mAP is only .0317 above the .447 floor. The team chose not to risk that narrow margin for the additional speed.
- **r320** has similar practice mAP to m640, but is slower because four inputs no longer share one model pass.
- **r384 and r512** trade throughput for more spatial detail and higher mAP. They are valid points on the curve, but do not match the selected speed objective.
- **r640** is the most accurate on practice, but the COCO1k result is almost the same as r512 while taking substantially longer.

> **Final choice:** m640 is not the fastest or most accurate model in isolation. It is the selected operating point on the measured Pareto frontier: 2.106 ms/photo and .5586 practice mAP@50 with the custom kernels and box-step v2 enabled.

Alternative detector architectures considered

We also checked whether replacing YOLOv4 would produce a better frontier. These are exploratory comparisons, not configurations used for the selected run. The alternative-model mAP figures below are fp32 CPU mAP@50 on COCO1k at 640 input size (YOLOv4 is shown with its card result); card timings are reported separately because the tested alternatives did not all complete the same end-to-end harness path.

Architecture	COCO1k mAP@50 at 640 (alternatives: fp32 CPU; YOLOv4: card)	Card evidence	Decision
YOLOv4	.7519 card base; .7524 kernels + v2 (r640)	Full selected pipeline and all six YOLOv4 configurations measured; m640 kernels + v2: 1.672 ms/photo, .6439 COCO1k.	Kept. The mature port plus custom kernels produced the best measured speed/accuracy frontier.
YOLOv8s	.640	640 card device path measured at 4.77 ms/frame before host decode/box work; 4 × 320 mosaic evaluation estimated about 1.19 ms/photo device time, not full harness time.	Rejected: lower accuracy than the YOLOv4 frontier; the mosaic COCO1k result was .537.
YOLO11s	.659	640 card device path measured at 6.20 ms/frame before host decode/box work; 4 × 320 mosaic evaluation estimated about 1.55 ms/photo device time, not full harness time.	Rejected: lower accuracy than the YOLOv4 frontier; the mosaic COCO1k result was .555.
YOLO11m	.710	Reference accuracy was evaluated; the card path was not rerun after discovering and fixing a port issue. The branch's roughly 10.5 ms figure was a claim, not a verified end-to-end result.	Not selected: below the YOLOv4 640 reference accuracy and no verified harness result to justify a switch.

> **Fair-comparison caveat:** the alternative-model card times exclude parts of the harness (including image decode and box processing), so they are lower bounds, not comparable end-to-end latency claims. The CPU-reference mAP figures and the YOLOv4 final pipeline figures also come from distinct evaluation paths. We used the alternatives to decide whether changing model families was promising, not to claim a precise head-to-head runtime win from incomplete runs.

Measurement note: Results are tied to the stated evaluation set, configuration and timing protocol. Per-kernel microbenchmarks, early pipeline ablations, practice-set end-to-end results, COCO1k comparisons and stress tests answer different questions; they should not be combined into a single additive speed-up calculation.

Project: YOLOv4 inference optimization on Tenstorrent Wormhole n150. Selected configuration: m640, custom kernels enabled, box-step v2 enabled.