What Does MLX 4-bit Cost? A Controlled Audit of Quantization for Code Generation on Apple Silicon

J. Melton
American Code Labs
jmelton@americancode.org

August 2026
Preprint.

Abstract

Developers running code models locally on Apple Silicon overwhelmingly use 4-bit weights from the `mlx-community` repository, yet no published figure states what that quantization costs on a code benchmark. Apple reports a single bf16-to-4-bit accuracy comparison, covering two models on one non-code benchmark; the conversion model cards carry no accuracy numbers at all. We measure the gap directly.

We evaluate five instruction-tuned code models from 1.5B to 8B parameters on HumanEval+, MBPP+, and a 26-task suite drawn from our own research codebase, comparing bf16 against 4-bit weights quantized from those same weights—one harness, one machine, one variable. Pooled across all five models and both benchmarks, 4-bit round-to-nearest loses to bf16 on 220 problems and wins on 125 (McNemar $p < 10^{-6}$), a mean cost of roughly 1 to 6 points that **shrinks as models grow** (+0.52 points per billion parameters): Phi-4-mini at 3.8B loses 6.5 points where Qwen2.5-Coder-7B loses 1.2.

Adding an activation-aware (AWQ) arm does not recover the loss. AWQ remains well below bf16 ($p = 8\times10^{-4}$) and is statistically indistinguishable from the uncalibrated default (133–116, $p = 0.31$), while costing up to 1h51m of quantization time for identical inference throughput. At these bit widths and scales, **the loss is a property of the bit width rather than of the algorithm that produces it.**

We also show that the intuitive way to estimate this cost—differencing a locally measured 4-bit score against a vendor's published bf16 figure—is unsound: on Qwen2.5-Coder-1.5B it reports −9.1 points where the controlled comparison gives −5.5, because the harness-mismatch term (−3.6) is the same order as the effect and does not even have a stable sign across benchmarks.

Completeness of the data. All 5 of 5 bf16-vs-4-bit pairs are measured, and 4 of 5 carry a calibrated AWQ arm (Phi-4-mini excluded—`mlx_lm.quant.awq` does not support its architecture). Every number below is measured and traceable to an EvalPlus per-problem results file; the cited figures are archived in `measured-results-v0313.json` (unified mlx-lm 0.31.3 runtime) and `measured-results.json` (the earlier 0.29.1 matrix, retained as a cross-version replication). **No cell is estimated, interpolated, or carried over from**

a different configuration. Where a measurement could not be made—Phi-4-mini's calibrated arm, and peak memory throughout—the text says so and states why, rather than substituting a plausible value.

1 Introduction

A developer who wants a code model running locally on a Mac has, in practice, one path: download a 4-bit conversion from the `mlx-community` repository and load it with `mlx-lm`. The weights are small enough to fit in unified memory, they load in seconds, and they generate roughly twice as fast as the original bf16 checkpoint. This is the artifact people actually run.

What it costs them in accuracy is not published anywhere we could find.

The vendor model cards for these conversions report no accuracy numbers; they are auto-generated conversion artifacts. Apple's own documentation contains exactly one bf16-to-quantized accuracy comparison—two models, on MMLU Pro, with no code benchmark [1]. The one MLX project that does evaluate HumanEval compares its output against another 4-bit configuration rather than against full precision, which measures a recipe difference rather than the cost of quantizing at all. The quantization literature offers a prior of roughly 0.5 to 2 points lost at 4 bits, but every study behind that figure used a *calibrated* method such as GPTQ [7] or AWQ [13], while MLX's default is uncalibrated round-to-nearest that additionally quantizes token embeddings and the output head—a configuration those papers did not measure.

1.1 Why the number is missing

The gap is not simply an oversight; it is structurally awkward to close, for three reasons we encountered rather than anticipated.

First, **the standard harness does not run on the relevant operating system.** EvalPlus [14] caps executed code via `RLIMIT_AS`, which Darwin rejects, so every problem errors out with a resource-limit exception. Every MLX user is on macOS. A one-line environment variable works around it (§3.2), but the default experience of trying to evaluate MLX weights with the field's standard tool is that nothing runs.

Second, **the obvious shortcut is wrong.** A practitioner who measures their 4-bit model locally and subtracts the model card's published bf16 score is not measuring quantization; they are measuring quantization plus every difference between their harness and the vendor's. On Qwen2.5-Coder-1.5B that shortcut reports a 9.1-point loss where the controlled comparison gives 5.5. The harness term is 3.6 points—the same order as the effect—and its sign is not stable: our harness scores below the published figure on HumanEval and above it on MBPP, so it cannot be corrected with a constant offset (§5).

Third, **the measurement is easy to get wrong quietly.** Building the harness for this study surfaced five independent defects, each of which scored *correct* model answers as failures, and none of which announced itself. The largest was a single undocumented protocol choice—whether the MBPP prompt shows the test assertion, and therefore conveys the function name the tests require—worth 59 points of pass@1. This is not incidental to the paper; it is why we abandoned our own harness for EvalPlus, and it corroborates

independent reports that only 12 of 35 published code-model results could be reproduced (§7).

1.2 What we do

We evaluate five instruction-tuned code models spanning 1.5B to 8B parameters on a single 32 GB Apple Silicon host. Each model is measured in bf16 and in 4-bit weights produced by `mlx_lm.convert` **from those same weights**, so the two arms differ in exactly one variable. Community uploads are deliberately not used as the 4-bit arm: their provenance cannot be verified, and we show in §6 that they are not equivalent to a local conversion. Four of the five models additionally receive a calibrated AWQ arm; the fifth cannot, because the tool does not support its architecture—and it is, awkwardly, the model with the largest quantization loss in the study.

Decoding is greedy with a single sample per problem. That choice requires determinism, which we verified rather than assumed: 40 of 40 completions were byte-identical across separate processes (§3.4). This localizes a prior result [17]—that temperature-zero decoding is non-deterministic in 47.6% of HumanEval tasks—as an artifact of API serving rather than a property of greedy decoding.

Alongside the public benchmarks we evaluate a 26-task suite derived from functions in our own research codebase. Public code benchmarks are documented as overwhelmingly easy—one audit finds 84.8% of HumanEval and 89.6% of MBPP problems fall in its lowest difficulty tier [25]—and they are demonstrably present in pretraining corpora. The domain suite is uncontaminated by construction and discriminates far more sharply across this model range: it spreads the five models by a factor of 5.5 where HumanEval spreads them by 1.3.

1.3 What we find

The cost of 4-bit is real, modest, and larger for smaller models. Pooled across all five models and both benchmarks, round-to-nearest loses 220 problems to bf16 and wins 125 ($p < 10^{-6}$). The per-model mean ranges from -1.2 to -6.5 points and regresses on parameter count at $+0.52$ points per billion—meaning the models most often chosen *because* they are small are the ones quantization hurts most.

Calibration does not fix this. AWQ stays well below bf16 ($p = 8 \times 10^{-4}$) and is indistinguishable from the uncalibrated default in a head-to-head that has remained at $p \approx 0.31$ across three successive additions of data. Its cost is paid entirely at build time—up to 1h51m of quantization for an 8B model—and inference throughput is identical. The useful conclusion for a practitioner is not "choose a better quantizer"; it is that at these bit widths and scales the loss belongs to the bit width itself.

2 Related Work

2.1 What HumanEval and MBPP actually measure

The benchmarks this study uses are known to be inadequate in two distinct ways, and both shape how our results should be read.

Test adequacy. EvalPlus [14] augments HumanEval with 80× more tests and MBPP with 35×, and finds pass@k falls by 19.3–28.9% across 26 models once the additional tests are applied. It also found 18 defects—roughly 11% of problems—in HumanEval's own reference solutions. We report both base and plus variants throughout for this reason, and use EvalPlus itself as the harness rather than a reimplementation (§3.2).

Contamination. Both benchmarks are demonstrably present in common pretraining corpora: 12.2% of HumanEval appears verbatim in The Pile and 18.9% in The Stack [15], with semantic overlap measured at 50.8% for MBPP and 63.4% for HumanEval against Stack samples [20]. Standard n-gram decontamination has low recall and is defeated by paraphrase [26]. A leak-free replacement benchmark scores models up to 43% lower than HumanEval.

This bounds what an absolute score means—but it is **largely neutralized by our design**. Both arms of every comparison are the same model on the same problems, so whatever contamination inflates the bf16 score inflates the 4-bit score too, and cancels in the paired difference. Contamination is a threat to the *level* of our numbers, not to the *deltas* that carry our claims.

Difficulty. PythonSaga's taxonomy audit [25] places 84.8% of HumanEval and 89.6% of MBPP problems in its lowest difficulty tier, with MBPP containing none in its highest. Harder or contamination-controlled alternatives consistently score models far lower: MHPP takes GPT-4o from 91.0% to 51.1% [4], EvoEval reports a 39.4% average drop across 51 models [24], and LiveCodeBench [10] shows the gap is concentrated in exactly the population we study—fine-tuned open models under ∼7B, where one model scores 59.8% on HumanEval+ but 26.3% on LiveCodeBench-Easy. This motivates our domain suite (§3.6) as a discriminating instrument rather than a replacement.

2.2 Quantization and code generation

The prior for 4-bit weight-only quantization is a loss of roughly 0.5 to 2 pass@1 points in the 1B–34B range. Giagnorio et al. [8] measure CodeLlama-7B at 29.8 → 29.1 on MultiPL-E Python going from FP16 to 4-bit with AQLM, with the sharp degradation appearing only at 3 bits and below. Kurtić et al. [12], across more than 500,000 evaluations, report W4A16 recovering 98.9% of BF16 HumanEval performance. Fang et al. [6], whose model range most closely matches ours, find 4-bit costs 1.19% of clean performance on average and conclude that smaller models tolerate quantization better than expected.

Not all results agree. Shi and Ding [22] report far larger degradation and argue coding and math are disproportionately fragile—though their catastrophic cases are dominated by activation quantization rather than the weight-only setting we study. Separately, scaling work finds low-bit degradation grows with training tokens and shrinks with model size [18], which places modern heavily-trained 1–8B models in the least favourable quadrant. **Our size trend (§4.1) is consistent with that prediction**; our absolute magnitudes sit at or slightly above the calibrated-method prior, which is what one would expect given that MLX's default is uncalibrated (§3.3).

Critically, **every study above measures a calibrated quantizer.** None measures naive round-to-nearest with embeddings and the output head also quantized, which is what

`mlx_lm.convert --q-bits 4` produces and what the artifacts on `mlx-community` are.

2.3 Apple Silicon evaluation

Published Apple Silicon LLM benchmarking is overwhelmingly about speed and memory rather than accuracy. Apple's own `mlx-lm` benchmark documentation [1] contains the only first-party bf16-to-quantized accuracy comparison we located: two models, on MMLU Pro, with a bf16→4-bit delta of −3.33 points for a 4B model. Crowdsourced Apple Silicon performance tables and recent throughput studies report tokens/sec and memory across many models and report no accuracy figures at all. The `mlx-community` conversion cards are auto-generated and carry none either.

We could find no published HumanEval or MBPP result for stock `mlx-community` 4-bit weights against a bf16 baseline, from Apple, from mlx-community, or from a third party. That intersection—MLX's default quantizer × code benchmarks × the 1–8B range—is the cell this paper fills.

2.4 Reproducibility of code-model results

Our harness findings (§7) sit in a small but consistent literature. Szalontai et al. [23] attempted to reproduce 35 published results on a HumanEval variant and succeeded on 12, attributing failures to benchmark-variant confusion, temperature misconfiguration, and precision that papers never state. Ouyang et al. [17] report that 47.56% of HumanEval tasks produce non-identical outputs across repeated requests even at temperature zero—a result we localize in §3.4 as an artifact of API serving rather than of greedy decoding. FormatSpread [21] demonstrates up to 76 accuracy points of spread from prompt formatting alone, albeit on classification rather than code.

What appears to be missing is a code-native account of how much the *harness* moves pass@1. §7 contributes five measured instances, the largest worth 59 points.

3 Methods

3.1 Design

One variable. For each model we evaluate two arms—bf16 and 4-bit—through an identical harness, on the same host, with identical decoding. Vendor published numbers are used **only** as a sanity check that our harness reproduces the standard protocol, never as a term in any difference.

3.2 Harness

EvalPlus 0.3.1 [5] (HumanEval+ 164 tasks, MBPP+ 378 tasks; the MBPP+ task count was revised from 399 in an earlier dataset version), rather than a bespoke harness. This is a deliberate reversal: we began with a custom implementation and abandoned it after finding five independent defects (§7).

EvalPlus does not run on macOS as shipped. `reliability_guard` caps executed code via `RLIMIT_AS`, which Darwin rejects, and every problem fails with `ValueError: current limit exceeds maximum limit`. The documented escape hatch is `EVALPLUS_MAX_-`

`MEMORY_BYTES=-1`. We note this because every MLX user is on macOS, and it plausibly contributes to the absence of published MLX code-benchmark numbers.

Generation runs under system Python 3.9 (where `mlx-lm` is installed) and scoring under Python 3.14 (where EvalPlus installs cleanly); EvalPlus decouples the two by design.

3.3 Quantization and provenance

4-bit arms are produced **locally** by `mlx_lm.convert --quantize --q-bits 4` from the same weights the bf16 arm evaluates, rather than downloaded from `mlx-community`. A community upload cannot be verified as to base revision or conversion settings, which would confound the single variable under test. §6 shows this is not a pedantic concern.

MLX's default quantizer is affine round-to-nearest [3]: per group of 64, scale and bias from min/max, no calibration data, no error compensation. Both `nn.Linear` and `nn.Embedding` define `to_quantized`, so **token embeddings and `lm_head` are quantized to 4 bits as well**—where GGUF Q4_K_M keeps them at Q6_K and GPTQ/AWQ pipelines conventionally leave them in fp16. Nominal "4-bit" is therefore not one thing.

3.4 Decoding and the determinism gate

Greedy, `temp=0.0`, $n = 1$, max 768 new tokens. $n = 1$ requires determinism, which we verified rather than assumed: 20 HumanEval problems generated twice within a process and twice across separate processes produced **40/40 byte-identical completions**.

This matters because Ouyang et al. [17] report that 47.6% of HumanEval tasks yield non-identical outputs at temperature 0. Our result localizes that finding: it is an artifact of API serving (batching, load balancing, kernel selection), not a property of greedy decoding. Local MLX greedy is bit-reproducible. Throughput varied 193–208 tok/s across those identical runs—speed is noisy, tokens are not.

3.5 Statistics

Arms are evaluated on identical problems, so observations are paired. We use McNemar's exact test on discordant pairs, not a difference of proportions. We report b (bf16 solved, 4-bit failed) and c (the reverse) throughout.

3.6 Domain task suite

Public benchmarks are saturated and skewed easy—PythonSaga finds 84.8% of HumanEval and 89.6% of MBPP problems are "Easy", with MBPP containing no "Hard" problems at all [25]. We therefore add 26 tasks derived from functions in our own research codebase (SAE training, circuit tracing, activation streaming, statistical nulls): 8 EASY, 8 MEDIUM, 10 HARD, 347 assertions.

Each task was validated in both directions: it must pass with a known-correct reference solution, **and** reject 21 hand-written plausible-but-wrong implementations. This two-sided validation caught one task whose stated trap was mathematically vacuous—it would have scored every model correct—and one that was unanswerable by construction.

3.7 Host

Apple Silicon, 32 GB unified memory. This ceiling excludes DeepSeek-Coder-V2-Lite from the bf16 arm (31.4 GB weights); it is reported 4-bit-only and contributes to no delta.

4 Results: Quantization Cost

4.1 Paired comparison

All five pairs are measured. Δ is the mean of the HumanEval and MBPP deltas; b and c are discordant pairs pooled over both benchmarks (bf16-only and 4-bit-only solves).

Table 1: Paired bf16-vs-4-bit comparison, all five models. Δ is in percentage points. † Qwen-3B's 4-bit arm is an `mlx-community` upload, not locally converted (§6); excluding it leaves 175 discordant pairs favouring bf16 against 102 favouring 4-bit, so neither the sign nor the significance of the pooled result changes.

Model	Params	HE bf16	HE 4bit	MBPP bf16	MBPP 4bit	mean Δ	b	c	p
Qwen2.5-Coder-1.5B	1.5B	0.671	0.616	0.704	0.680	−3.9	35	17	**0.018**
Qwen2.5-Coder-3B†	3.0B	0.841	0.811	0.772	0.728	−3.8	45	23	**0.010**
Phi-4-mini	3.8B	0.665	0.616	0.646	0.563	**−6.5**	91	52	**0.0014**
Qwen2.5-Coder-7B	7.0B	0.896	0.878	0.849	0.844	−1.2	16	11	0.442
Granite-8B-Code	8.0B	0.591	0.579	0.672	0.648	−1.8	33	22	0.177
All five pooled							**220**	**125**	$< \mathbf{10^{-6}}$

Two results, not one.

1. **The cost is real.** 220 discordant pairs favour bf16 against 125 favouring 4-bit, $p < 10^{-6}$. Individually only three of five models reach significance, which is why single-model studies of this effect are underpowered—an important observation in its own right.

2. **The cost shrinks with model size.** Regressing mean Δ on parameter count gives +0.52 **points per billion**: Phi-4-mini (3.8B) loses 6.5 points while Qwen-7B loses 1.2. This is consistent with the quantization literature's finding that larger models are more resilient [18], and it is the practically actionable result—the models most often chosen *because* they are small are the ones 4-bit hurts most.

Phi-4-mini is the outlier in both directions—the largest quantization cost in the matrix (−6.5) and the largest gap to its published bf16 figure (−7.9, §4.2). We do not have an explanation. Two candidates are consistent with the evidence but untested here: its published figure comes from an internal pipeline using 3-shot MBPP where we use 0-shot [16], which would inflate the reported baseline without affecting our paired delta; and MLX quantizes token embeddings and the output head (§3.3), so a model whose embedding matrix is unusually large relative to its body would be disproportionately affected. Distinguishing these would require an arm with embeddings held at higher precision, which `mlx_lm.convert` supports only through a non-default mixed recipe. We flag it rather than resolve it.

4.2 bf16 baselines against published figures

Sanity check only—never differenced against a 4-bit score. Table 2 gives our bf16 HumanEval scores beside the published figures.

Table 2: Our bf16 HumanEval scores against published figures. ‡ IBM publishes HumanEval-Synthesize (57.9 Python) rather than vanilla HumanEval [9]; our 59.1 is near it but not the same benchmark.

Model	Ours (bf16)	Published	Δ
Qwen2.5-Coder-1.5B	0.671	0.707	−3.6
Qwen2.5-Coder-3B	0.841	0.841	**0.0**
Qwen2.5-Coder-7B	0.896	0.884	+1.2
Phi-4-mini	0.665	0.744	−7.9
Granite-8B-Code	0.591	n/a‡	—

Agreement is close across the Qwen family [19] and exact at 3B, which is the strongest available evidence that the harness reproduces the vendor protocol. Phi-4-mini's larger gap is consistent with Microsoft evaluating through an internal pipeline whose prompt format and shot count are unspecified.

4.3 Throughput

Table 3 reports generation throughput across all three arms.

Table 3: Generation throughput (tok/s) during the HumanEval sweep, all arms, under the unified mlx-lm 0.31.3 runtime. Speedup is 4-bit RTN over bf16.

Model	bf16	4-bit RTN	4-bit AWQ	speedup
Qwen2.5-Coder-1.5B	90.6	164.8	164.8	1.82×
Qwen2.5-Coder-3B	48.1	97.8	97.5	2.03×
Phi-4-mini	41.0	88.9	n/a	2.17×
Qwen2.5-Coder-7B	22.8	57.3	57.2	2.51×
Granite-8B-Code	19.7	48.3	48.2	2.45×

4-bit is 1.8–2.5× faster, and **the speedup grows with model size**—the opposite of the accuracy cost, which shrinks with size (§4.1). The two trends compound in the same direction: larger models gain more throughput from quantization and lose less accuracy to it.

RTN and AWQ are throughput-identical to within measurement noise (164.8 vs 164.8; 57.3 vs 57.2; 48.3 vs 48.2). Whatever the calibrated quantizer costs is paid entirely at build time (§4.4), never at inference.

Memory was not measured reliably, and no memory claim is made. Our collection path reports 4-bit consuming *more* peak memory than bf16 on Qwen-3B (6.55 vs 6.47 GB) while being sane on Qwen-1.5B (1.65 vs 3.48 GB)—an internal contradiction we

did not resolve. Since the memory reduction is half the practical motivation for 4-bit, this is a substantive limitation rather than a footnote, and is restated in §10. Readers should take the well-established weight-footprint reduction (∼4× on the quantized tensors) as the basis for memory reasoning, not any figure from this study.

4.4 Calibration does not rescue 4-bit

MLX's default quantizer is uncalibrated round-to-nearest (§3.3), which is weaker than the GPTQ/AWQ-class methods behind the published ∼0.5–2 pt loss figures. The obvious hypothesis is that the cost we measure is an artifact of that default rather than a property of 4-bit. We tested it directly by adding an activation-aware (AWQ) arm [13] at the same bit width and group size.

Table 4 gives the three-arm comparison. All three arms ran under a single mlx-lm 0.31.3 (see §4.5 on why the runtime changed and why it is not a confound). **Four of five models carry a calibrated arm; Phi-4-mini cannot (‡).**

Table 4: Three-arm comparison. Δ columns are percentage points against bf16.

Model	Benchmark	bf16	4-bit RTN	4-bit AWQ	RTN Δ	AWQ Δ
Qwen2.5-Coder-1.5B	HumanEval	0.683	0.622	0.628	−6.1	−5.5
	MBPP	0.704	0.688	0.683	−1.6	−2.1
Qwen2.5-Coder-3B	HumanEval	0.854	0.829	0.835	−2.4	−1.8
	MBPP	0.775	0.712	0.738	−6.3	−3.7
Granite-8B-Code	HumanEval	0.598	0.585	0.549	−1.2	−4.9
	MBPP	0.677	0.648	**0.680**	−2.9	**+0.3**
Qwen2.5-Coder-7B	HumanEval	0.896	0.878	0.890	−1.8	−0.6
	MBPP	0.854	0.844	0.841	−1.1	−1.3
Phi-4-mini		0.665	0.616	**n/a**‡	−4.9	

‡ **Phi-4-mini cannot have an AWQ arm.** `mlx_lm.quant.awq` raises `NotImplementedError: AWQ support for phi3 models NYI`. This is a hard limitation of the tool, not a transient failure. It matters because Phi-4-mini carries the *largest* RTN cost in the matrix (§4.1), so it is precisely the model where "a better quantizer would fix this" was most plausible—and it is the one model where we cannot test it. Reported as excluded with cause rather than silently dropped.

Table 5 pools McNemar over the 8 cells where all three arms exist (4 models × 2 benchmarks; Phi-4-mini is absent because it has no AWQ arm). These counts are therefore a subset of the five-model totals in §4.1 and are not directly comparable to them.

Table 5: Pooled McNemar over the 8 three-arm cells (4 models × 2 benchmarks).

Comparison	wins A	wins B	p
bf16 vs RTN	134 (bf16)	70 (RTN)	**0.000009**
bf16 vs AWQ	119 (bf16)	72 (AWQ)	**0.00083**
RTN vs AWQ (head-to-head)	133 (AWQ)	116 (RTN)	**0.31**

Both quantizers lose to bf16 by a wide statistical margin. Neither beats the

other. The head-to-head has been stable at $p \approx 0.31$ across three successive additions of data, and Qwen-7B's own head-to-head is 14–13 ($p = 1.00$)—as close to a tie as the test can express.

The practical conclusion is therefore not "use a better quantizer". It is that at 4 bits and these scales **the loss is a property of the bit width, not of the algorithm that gets you there.**

Per-benchmark variance exceeds the method difference. Granite-8B is the clearest case: AWQ is 4.9 points *below* bf16 on HumanEval but *matches* it on MBPP (+0.3), while RTN loses on both. On the same model with the same weights, the two benchmarks disagree about which quantizer is better. This is the section's central caution. Read one benchmark on one model and the directional conclusion reverses—the same failure this paper documents in §7, arriving from a different direction. **A single benchmark on a single model does not support a directional claim about quantizer quality.**

The head-to-head also bears on §6, where an `mlx-community` upload outscored a local RTN conversion and appeared to suggest that recipe quality matters. AWQ versus RTN is a far larger recipe difference, measured on four models, and it finds no significant effect ($p = 0.31$). Both can be true—the community uploads may differ from local `convert` in some way other than calibration—but the simpler reading is that §6's effect is noise, and it is reported as non-significant there.

Cost of the calibrated arm. AWQ quantization scales worse than linearly: 18 minutes at 1.5B, ∼1h30m at 7B, 1h51m at 8B—roughly 6× the time for 5.3× the parameters, as the grid search over scaling factors compounds with layer count. Inference throughput is unchanged (Granite-8B 48.2 vs 48.3 tok/s; Qwen-7B 57.2 vs 57.3), so the price is paid entirely at build time and buys no runtime benefit either.

Domain suite. The 26-task suite tracks the same ordering but with a wider spread, and is the one place AWQ looks consistently worse (Table 6).

Table 6: Domain task suite, three arms, tasks solved of 26.

Model	bf16	RTN	AWQ
Qwen-7B	12/26	11/26	**8/26**
Granite-8B	7/26	6/26	7/26

Qwen-7B's AWQ arm solves four fewer tasks than bf16 while its HumanEval score is within 0.6 points. With only 26 tasks this is a small-N observation, not a result—but it is the direction worth checking if the suite is expanded, since it hints that aggregate pass@1 on saturated benchmarks may hide capability loss the harder tasks expose.

One asymmetry to record rather than smooth over. AWQ sets `model.embed_tokens` to `group_size: 32` while RTN uses 64 uniformly. So "RTN vs AWQ" is not a pure method comparison—the embedding treatment differs too. Given §3.3 (MLX quantizes embeddings and `lm_head` at all, unlike GGUF Q4_K_M and GPTQ/AWQ pipelines elsewhere), this is a plausible contributor and should not be attributed to calibration alone.

4.5 Runtime version is not a confound

The calibrated quantizers exist only in mlx-lm ≥ 0.30 [2], while the matrix in §4.1 was measured under 0.29.1. The original plan was to produce AWQ weights with a newer mlx-lm in an isolated venv and evaluate them on the pinned runtime, keeping inference constant. That failed: 0.31.3 writes a per-layer quantization config that 0.29.1 cannot parse. We rejected writing a compatibility shim, because misreading a quantization spec would produce an AWQ column that looks correct and means something else.

Instead we tested the assumption being protected (Table 7). Same weights, same harness, only the mlx-lm version differing.

Table 7: Runtime-version control on Qwen-1.5B. 162 of 164 per-problem verdicts are identical.

	0.29.1	0.31.3	Δ	discordant	p
Qwen-1.5B HumanEval	0.671	0.683	+1.2	0 old-only / 2 new-only	0.50
Qwen-1.5B HumanEval+	0.622	0.634	+1.2	0 / 2	0.50

162 of 164 per-problem verdicts are identical. The runtime shifts 2 problems; the effect under study accounts for 345 discordant pairs across the matrix. The version is therefore not a confound at the resolution that matters, and the three-arm results run under a single 0.31.3.

The 0.29.1 matrix is retained and reported (§4.1) rather than discarded—it is a cross-version replication, which is more than the design originally promised. Every summary records `mlx_lm_version`, `python`, and `runs_base` so no score can be silently attributed to the wrong inference stack.

5 Results: The Vendor-Subtraction Confound

The intuitive estimate of quantization cost—measure 4-bit locally, subtract the model card's bf16 figure—is available to any practitioner and is wrong (Table 8).

Table 8: Two estimates of the same quantity, Qwen-1.5B HumanEval.

Method	Implied cost
Naive: published bf16 (0.707) − measured 4-bit (0.616)	**−9.1** pts
Controlled: measured bf16 (0.671) − measured 4-bit (0.616)	**−5.5** pts

The naive figure absorbs the entire harness-mismatch term (−3.6 pts here), which is of the same order as the effect. The sign of that term is not even stable: our harness scores *below* published on HumanEval for Qwen-1.5B but *above* published on MBPP, ruling out a simple "our prompt is worse" explanation and showing the bias cannot be corrected with a constant.

The same comparison for Qwen2.5-Coder-3B, whose 4-bit arm is provenance-matched in the unified run: published bf16 0.841, measured bf16 0.854, measured 4-bit 0.829. The naive estimate gives −1.2 and the controlled one −2.5—here the shortcut *understates* the

cost, because our harness scores above the published figure on this model. The error does not have a consistent direction, which is precisely why it cannot be corrected for.

6 Observation: Community Uploads May Not Equal Local Conversions

This section reports an observation, not a result. It is retained because it motivates a methodological choice (§3.3) and because the alternative—omitting a measurement that did not reach significance—is the kind of selective reporting this paper argues against elsewhere.

Early in this study the 4-bit arm was an `mlx-community` upload rather than a local conversion. Measuring both for Qwen2.5-Coder-1.5B, evaluated identically (Table 9):

Table 9: Community upload against local conversion, Qwen2.5-Coder-1.5B.

Benchmark	community upload	self-converted	Δ	p
HumanEval	0.646	0.616	−3.0	0.18
MBPP	0.690	0.680	−1.1	0.46
pooled				**0.11**

The upload scores higher on both benchmarks, but neither cell nor the pooled comparison reaches significance on a single model.

Why we do not claim this. Two things argue against promoting it. First, it was never replicated: we switched to local conversion for provenance control (§3.3) and did not measure a second community upload, so this rests on one model where $p = 0.11$. Second, it sits in tension with §4.4. This observation suggests the conversion recipe matters; the AWQ head-to-head—a far larger recipe difference, on four models—found no significant effect ($p = 0.31$). The simplest reading consistent with both is that this section's difference is noise.

What it does justify is the methodological choice. Whether or not the difference is real, a community upload's provenance cannot be verified: the base revision and conversion settings are not recorded on the auto-generated cards. Using one as the 4-bit arm would place an unverifiable artifact on one side of a comparison whose entire value rests on differing in a single variable. Converting locally removes that risk at no cost, and we recommend it for any study of this kind.

If someone wants to settle it, the experiment is cheap: evaluate the community upload and a local conversion of the same base weights across several models. It is worth settling, because if uploads really are better then every number in this paper is a lower bound on what practitioners actually deploy—they run the uploads, not local conversions.

7 Harness Fragility as a Finding

Five independent defects were found in a custom HumanEval/MBPP harness before it was abandoned (Table 10). Every one silently scored **correct** model answers as failures.

Table 10: Five harness defects, each of which scored correct answers as failures.

Defect	Measured effect
MBPP prompt never conveyed the required function name, which `test_list` asserts exactly	MBPP pass@1 **0.047** → **0.640**
One system prompt demanded a bare body—correct for HumanEval's stub, fatal for MBPP which has none	SyntaxError on every compliant answer
`test_imports` never emitted	13/257 unconditional NameErrors
Unindented body appended to a HumanEval stub	SyntaxError on correct answers
Extractor's non-greedy fence match truncated any answer containing a code-fence delimiter	one task unanswerable by construction

The first defect is the headline: a single undocumented protocol choice moved pass@1 by 59 points. EvalPlus's own MBPP+ prompts include the assertion, so this was a deviation from the accepted protocol rather than a hard benchmark. This corroborates Szalontai et al. [23], who reproduced only 12 of 35 published results and attributed failures to benchmark-variant confusion, temperature misconfiguration, and unspecified precision.

The practical rule this suggests: **a benchmark number is not evidence until the harness has been shown to score a known-correct reference answer as passing.** Our domain suite enforces this in both directions (§3.6).

These belong in the body rather than an appendix. The reproducibility literature establishes that harness configuration moves scores—Szalontai et al. [23] reproduced 12 of 35 published results, FormatSpread [21] measured 76 points of spread from formatting—but those are, respectively, a variant benchmark and a classification task. We could find no code-native measurement of how much the harness itself moves pass@1. The first defect is exactly that measurement: 59 points, from a protocol choice that papers do not document. A result of that magnitude, in the same units as every score this paper reports, is not supporting material.

8 Results: Domain Task Suite

Table 11 sets the domain suite beside HumanEval for the bf16 arms.

The suite spreads this model range 5.5× (2 → 11) where HumanEval spreads it 1.3× (0.671 → 0.896). The best model solves 42% of tasks; the smallest solves 8%. This is consistent with the PythonSaga finding that public benchmarks are dominated by easy problems [25], and supports using an uncontaminated domain suite as the discriminating instrument.

A per-tier and per-trap-family breakdown across arms is possible from the recorded per-task results, but at 26 tasks split three ways the cells are too small to support inference—a single task moves a tier by 10–12 percentage points. We report the aggregate only, and note that expanding the suite is the prerequisite for any finer-grained claim (§9.5).

Table 11: Domain task suite against HumanEval, bf16 arms. DeepSeek-V2-Lite is 4-bit-only (its bf16 weights exceed the host's 32 GB) and contributes to no delta.

Model	Domain suite	HumanEval
Qwen2.5-Coder-1.5B	2/26	0.671
Qwen2.5-Coder-3B	5/26	0.841
Granite-8B-Code	6/26	0.591
Phi-4-mini	7/26	0.665
DeepSeek-V2-Lite (4-bit)	7/26	0.762
Qwen2.5-Coder-7B	**11/26**	0.896

9 Discussion

9.1 What this means if you are choosing weights

For most local coding work, **4-bit is the right default, and the reason is the exchange rate rather than the loss being negligible.** The loss is real and statistically unambiguous ($p < 10^{-6}$ pooled), but it buys roughly a doubling of throughput and roughly half the memory. On a 32 GB machine that trade is often what makes a model usable at all: Qwen2.5-Coder-7B in bf16 leaves little headroom, while its 4-bit conversion runs comfortably alongside an editor and a browser.

The more actionable finding is **where the cost falls.** It shrinks as models grow—+0.52 points per billion parameters—so the penalty is largest exactly where users are most likely to accept it. Phi-4-mini at 3.8B loses 6.5 points; Qwen2.5-Coder-7B loses 1.2. A practitioner choosing a small model *because* it is small is paying the steepest quantization tax, and would often be better served by a larger model at 4 bits than a smaller one at higher precision—a conclusion consistent with prior work finding that at equal memory, bigger-model-lower-bit beats smaller-model-higher-bit [11].

9.2 "Use a better quantizer" is not the fix

The intuitive response to a quantization loss is to reach for a better quantization method. Our data does not support that move. AWQ—activation-aware, calibrated, the class of method behind the published ∼1-point figures—remains well below bf16 ($p = 8 \times 10^{-4}$) and is statistically indistinguishable from the naive default (133–116, $p = 0.31$). That head-to-head held at $p \approx 0.31$ across three successive additions of data, so it is stable rather than merely underpowered, and Qwen-7B's own head-to-head is 14–13.

The cost side makes this decisive rather than merely inconclusive. AWQ quantization took up to 1h51m for an 8B model, scaling worse than linearly with parameter count, and produced **identical inference throughput** (57.2 vs 57.3 tok/s). A practitioner would be spending hours of compute for a benefit this study cannot detect.

We state the conclusion as: at 4 bits and these scales, **the loss is a property of the bit width, not of the algorithm that produces it.** We are careful not to overreach. This is one calibrated method on four models; GPTQ and DWQ are unmeasured, and AWQ and RTN differ in embedding group size (32 vs 64) as well as in calibration, so the head-to-head is not a pure method contrast.

9.3 Do not compare your numbers to a model card

The single most transferable result here is negative. Differencing a locally measured 4-bit score against a vendor's published bf16 figure is a natural thing to do and it does not work: on Qwen2.5-Coder-1.5B it reports −9.1 points where the controlled comparison gives −5.5. The harness-mismatch term is 3.6 points—the same order as the effect—and its sign is not stable across benchmarks, so it cannot be absorbed into a correction factor.

The corollary is that a bf16 baseline is not optional. It is the majority of the experimental cost in this study, and it is the only part that makes the 4-bit number mean anything.

9.4 The measurement is the hard part

Five of the defects we found (§7) scored *correct* answers as failures, and none announced itself. The largest—whether the MBPP prompt conveys the function name its tests assert—was worth 59 points of pass@1, comfortably larger than every effect this paper reports. That we found it only because a 4.7% score was too implausible to accept is not reassuring; a defect of half that size would have produced a believable number and been published.

This shaped our practice in ways we would recommend generally. Every task in the domain suite must pass with a known-correct reference **and** reject 21 hand-written wrong implementations, a two-sided check that caught one task whose trap was mathematically vacuous (it would have scored every model correct) and one that was unanswerable by construction. Scores are read from the harness's own per-problem verdict files rather than from any summary a runner wrote. And a benchmark sweep that returns 0/N is treated as a harness fault rather than a result.

The same discipline applies to interim results. A directional conclusion drawn from Granite-8B's HumanEval cell alone—that calibration "actively hurts"—was reversed by its MBPP cell within the hour. Per-benchmark variance on a single model exceeded the method difference being characterized.

9.5 What we would measure next

Three directions follow directly. **A second calibrated method** (GPTQ or DWQ) would test whether "calibration does not help" generalizes beyond AWQ, though given the stability of the head-to-head we would expect it to strengthen rather than change the conclusion. **An expanded domain suite** would resolve the study's most intriguing loose end: Qwen-7B's AWQ arm solves 8 of 26 domain tasks against RTN's 11 and bf16's 12, while its HumanEval scores differ by 0.6 points—hinting that aggregate pass@1 on saturated benchmarks may conceal capability loss that harder tasks expose. At $n = 26$ that is an observation, not a result. **Mixture-of-experts models** are a genuinely different question: sparse activation and expert routing interact with quantization in ways dense-model results cannot predict, and the current generation of strong open coding models is increasingly MoE.

10 Threats to Validity

- **Provenance mismatch on Qwen-3B** (Table 1†).
- **Calibrated arm is partial.** AWQ is measured on 4 of 5 models (§4.4)—Phi-4-mini is excluded because `mlx_lm.quant.awq` does not support its architecture. GPTQ and

DWQ remain unmeasured; the head-to-head is RTN vs AWQ only, so "calibration does not help" is supported for one calibrated method, not for calibration in general.

- **AWQ and RTN differ in embedding group size** (32 vs 64), so §4.4's head-to-head is not a pure method contrast.
- $n = 1$ **greedy.** Justified by the determinism gate (§3.4), but a single sample per problem cannot estimate pass@k.
- **Single host.** One machine, one MLX version. No claim about other Apple Silicon configurations.
- **Memory figures unusable** (§4.3).
- **Data collection was not first-time-clean.** Three arms of the matrix failed twice before completing, for unrelated infrastructure reasons—a runner bug that pre-created the conversion output directory (which `mlx_lm convert` rejects), and a task daemon that silently accepted work without persisting it. All three were subsequently re-run and are included in the results above. Neither failure affects the validity of any measured number, but both are recorded because a silent half-empty matrix is the failure mode this kind of audit is most exposed to.

Data Availability

Per-problem EvalPlus verdict files, the archived result matrices (`measured-results-v0313.json` for the unified mlx-lm 0.31.3 runtime and `measured-results.json` for the 0.29.1 cross-version replication), the domain task suite with its reference solutions and 21 negative implementations, and the conversion and evaluation scripts are available at

https://github.com/american-code/mlx-quantization-audit

References

[1] Apple. mlx-lm benchmarks, 2025. URL https://github.com/ml-explore/mlx-lm/blob/main/mlx_lm/BENCHMARKS.md. The only first-party bf16-vs-quantized accuracy comparison located: two models, MMLU Pro, −3.33 points bf16→q4 for a 4B model.

[2] Apple. mlx-lm learned quantization, 2025. URL https://github.com/ml-explore/mlx-lm/blob/main/mlx_lm/LEARNED_QUANTS.md. AWQ, GPTQ, DWQ and dynamic-quant CLIs; available only in mlx-lm $\geq$ 0.30.

[3] Apple. `mlx.core.quantize` documentation, 2025. URL https://ml-explore.github.io/mlx/build/html/python/_autosummary/mlx.core.quantize.html. Affine round-to-nearest, per-group scale and bias.

[4] Jianbo Dai et al. MHPP: Exploring the capabilities and limitations of language models beyond basic code generation. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2405.11430. GPT-4o 91.0% → 51.1%.

[5] EvalPlus contributors. EvalPlus project repository, 2024. URL https://github.com/evalplus/evalplus. Dataset versions and the MBPP+ task-count revision (399 → 378).

[6] Sen Fang, Weiyuan Ding, Antonio Mastropaolo, and Bowen Xu. Smaller = weaker? benchmarking robustness of quantized LLMs in code generation. *arXiv preprint arXiv:2506.22776*, 2025. URL https://arxiv.org/abs/2506.22776. Closest model range to this study; 4-bit costs 1.19% of clean performance on average.

[7] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2023. URL https://arxiv.org/abs/2210.17323.

[8] Alessandro Giagnorio, Antonio Mastropaolo, Saima Afrin, Massimiliano Di Penta, and Gabriele Bavota. Evaluating the impact of post-training quantization on the performance of code-generation LLMs. *Empirical Software Engineering*, 2025. URL https://arxiv.org/abs/2503.07103. AQLM; CodeLlama-7B MultiPL-E Python 29.8 (FP16) → 29.1 (4-bit); degradation cliff at 3 bits.

[9] IBM. Granite code models: A family of open foundation models for code intelligence. *arXiv preprint arXiv:2405.04324*, 2024. URL https://arxiv.org/abs/2405.04324. Reports HumanEvalSynthesize rather than vanilla HumanEval.

[10] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination-free evaluation of large language models for code. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2403.07974. Contamination-free time-windowed evaluation; the 59.8% HumanEval+ vs 26.3% LCB-Easy gap for a fine-tuned sub-2B model.

[11] Renren Jin et al. A comprehensive evaluation of quantization strategies for large language models. *arXiv preprint arXiv:2402.16775*, 2024. URL https://arxiv.org/abs/2402.16775. At equal memory, larger-model-lower-bit outperforms smaller-model-higher-bit.

[12] Eldar Kurtić, Alexandre Marques, Shubhra Pandit, Mark Kurtz, and Dan Alistarh. "give me BF16 or give me death"? accuracy-performance trade-offs in LLM quantization. In *Proceedings of the Association for Computational Linguistics (ACL)*, 2025. URL https://arxiv.org/abs/2411.02355. More than 500,000 evaluations; W4A16 recovers 98.9% of BF16 HumanEval.

[13] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. AWQ: Activation-aware weight quantization for LLM compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2024. URL https://arxiv.org/abs/2306.00978. The method underlying our calibrated arm.

[14] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems (NeurIPS)*,

2023. URL https://arxiv.org/abs/2305.01210. EvalPlus; 80×/35× test augmentation, 19.3–28.9% pass@k reduction, 18 defective HumanEval reference solutions. Used as this study's harness.

[15] Alexandre Matton, Tom Sherborne, Dennis Aumiller, et al. On leakage of code generation evaluation datasets. In *Findings of the Association for Computational Linguistics: EMNLP*, 2024. URL https://arxiv.org/abs/2407.07565. 12.2% of HumanEval in The Pile, 18.9% in The Stack; leak-free LBPP scores up to 43% lower.

[16] Microsoft. Phi-4-Mini technical report. *arXiv preprint arXiv:2503.01743*, 2025. URL https://arxiv.org/abs/2503.01743. HumanEval 0-shot / MBPP 3-shot via an internal pipeline.

[17] Shuyin Ouyang, Jie M. Zhang, Mark Harman, and Meng Wang. An empirical study of the non-determinism of ChatGPT in code generation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2025. URL https://arxiv.org/abs/2308.02828. 47.56% of HumanEval tasks non-identical at temperature 0.

[18] Xu Ouyang et al. Low-bit quantization favors undertrained LLMs: Scaling laws for quantized LLMs with 100t training tokens. *arXiv preprint arXiv:2411.17691*, 2024. URL https://arxiv.org/abs/2411.17691. Degradation grows with training tokens and shrinks with model size.

[19] Qwen Team. Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*, 2024. URL https://arxiv.org/abs/2409.12186. Published HumanEval/MBPP figures used as the sanity check (Table 16, EvalPlus, 0-shot).

[20] Martin Riddell, Ansong Ni, and Arman Cohan. Quantifying contamination in evaluating code generation capabilities of language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 14116–14137, 2024. URL https://aclanthology.org/2024.acl-long.761/. 50.8% (MBPP) and 63.4% (HumanEval) semantic overlap with Stack samples.

[21] Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. Quantifying language models' sensitivity to spurious features in prompt design. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2310.11324. FormatSpread; up to 76 accuracy points of spread from formatting alone.

[22] Tianyao Shi and Yi Ding. Systematic characterization of LLM quantization: Performance, energy, and quality. *arXiv preprint arXiv:2508.16712*, 2025. URL https://arxiv.org/abs/2508.16712. The dissenting result; large coding degradation, dominated by activation quantization rather than the weight-only setting studied here.

[23] Balázs Szalontai, Balázs Márton, Balázs Pintér, and Tibor Gregorics. Investigating reproducibility challenges in LLM bugfixing on the HumanEvalFix benchmark. *Software (MDPI)*, 4(3):17, 2025. URL https://doi.org/10.3390/software4030017. 12 of 35 published results reproduced; causes include benchmark-variant confusion and unspecified precision.

[24] Chunqiu Steven Xia, Yinlin Deng, and Lingming Zhang. Top leaderboard ranking = top coding proficiency, always? EvoEval. *arXiv preprint arXiv:2403.19114*, 2024. URL https://arxiv.org/abs/2403.19114. 39.4% average drop across 51 LLMs on evolved problems.

[25] Ankit Yadav and Mayank Singh. PythonSaga: Redefining the benchmark to evaluate code generating LLMs. *arXiv preprint arXiv:2401.03855*, 2024. URL https://arxiv.org/abs/2401.03855. Difficulty taxonomy; 84.8% of HumanEval and 89.6% of MBPP in the lowest tier.

[26] Shuo Yang, Wei-Lin Chiang, Lianmin Zheng, Joseph E. Gonzalez, and Ion Stoica. Rethinking benchmark and contamination for language models with rephrased samples. *arXiv preprint arXiv:2311.04850*, 2023. URL https://arxiv.org/abs/2311.04850. n-gram decontamination defeated by paraphrase.