# QKERNELBENCH: BENCHMARKING LLMS FOR QUANTIZED GPU KERNEL GENERATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Quantized GPU kernels are central to efficient LLM inference, yet kernel benchmarks judge them as they judge dense kernels: by comparing a candidate's outputs with a reference's under a tolerance. A quantizer that computes its own scale returns quantized values (codes) together with that scale, and the codes must be judged together with it: the same codes can be correct under a scale one ULP from the reference's and wrong under the reference's own. A check that compares codes and scale with the reference's separately rejects correct kernels, accepts wrong ones, or both. We introduce QKERNELBENCH, a benchmark of 84 CUDA kernel contracts for quantized inference. Each contract states its numerical semantics clause by clause, and its verdict follows from them: codes are accepted when the returned scale is admissible and a declared arithmetic path reproduces them bit for bit, other codes must be exact outside declared one-step masks, and floating-point outputs must lie within declared bounds: a number of representable steps, a bound derived from the declared accumulation, or a tolerance. Every verdict is calibrated against mutants of its own contract and reports the positions it rejects. We judge kernels written by eight LLMs in two rounds, the second generated after the contracts were frozen. Frontier models solve 71 to 80 of the 84 contracts in the first round, but their kernels reach only a fraction of the speed of the shipping kernels we timed. On the same kernels, the correctness rules of published benchmarks err in both directions: they reject kernels whose scale differs lawfully from the reference's and accept kernels that break a stated clause, and no distance threshold separates the two. Under KernelBench's `allclose`, each frontier model would be credited with 14 to 18 fewer contracts. After repairs that the shipping kernels of eight production libraries prompted, the contracts accept their scale paths, reassociation and tensor-core accumulation; the kernels still rejected follow another convention for a stated clause, such as rounding, saturation or the zero guard, compute in a lower precision than the contract admits, or return codes that change between calls.

## 1 INTRODUCTION

Serving large language models at acceptable cost depends on quantization (Frantar et al., 2022; Lin et al., 2024; Dettmers et al., 2022; Micikevicius et al., 2022). Its memory and throughput gains are realized only through specialized kernels that pack sub-byte data, drive INT8 and FP8 tensor cores with the right accumulator precision, and fuse dequantization into a GEMM main loop. Reported LLM pass rates on such kernels are low. On the six quantization tasks of KernelBench-X, none of five methods succeeds (0 of 30) (Wang et al., 2026), and a benchmark built from serving traces reports low pass rates on FP8 and FP4 workloads (Xing et al., 2026).

What a pass rate measures depends on the rule that decides correctness, and current rules compare outputs under a tolerance. A quantizer, however, returns quantized values, the INT8, INT4, FP8 or FP4 bit patterns we call codes, together with the scale that interprets them. Two kernels can differ from the reference's codes at the same positions and by the same amount, one because it computed a lawfully different scale and the other because it is wrong, and a tolerance on the codes cannot tell them apart. The rule changes what a benchmark reports: judged by KernelBench's `allclose` on every case, each frontier model we evaluate would be credited with 14 to 18 fewer of our 84



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

contracts than the contract credits it, in each of two rounds, more than the frontier models differ from one another (Section 5.2).

**Correct outputs that the rules reject.** A tolerance on the codes can reject a correct kernel. `fp8_per_token_act_quant` quantizes each token to FP8 E4M3 codes and returns the per-token scale with them. Each of the six frontier models wrote a kernel that computes the scale as the prompt states it, $\mathrm{absmax}/448$ in `float32`; the PyTorch reference, whose division by a constant multiplies by its reciprocal on the GPU, lands one step away in 71 of the 128 rows of its own case, a difference the contract allows, and codes next to a rounding boundary move by one step. At the scale each kernel returns every code is reproduced bit for bit, and the contract accepts all six on every case at three seeds. Near the top of the E4M3 grid one step is $32.0$ in value, far outside the tolerance of KernelBench's `allclose`, and FlashInfer-Bench's matched-ratio rule at its shipped default ratio of $1.0$ admits no element outside its tolerance, so both reject all six correct kernels, and SGLang's shipping kernel for the same reason (Appendix G).

**A wrong kernel that the rules accept.** The opposite failure is a violation that no distance can see. `e2m1_decode` maps a 4-bit FP4 code to its value, and its contract states that code 8 decodes to $-0.0$. Asked once with the contract as its prompt, Claude Sonnet 5 wrote a kernel that decodes it to $+0.0$, so its output differs from the reference's in the sign bit at 16,333 of 262,144 positions. IEEE 754 defines $-0.0$ and $+0.0$ as equal, so every elementwise difference is zero and all five published rule settings we evaluate accept the kernel; the contract compares bit patterns, as the task requires, and rejects it. The sign changes no sum, and TensorRT-LLM's MXFP4 dequantizer makes the same choice, but the contract states $-0.0$, as the OCP microscaling format does (Open Compute Project, 2023), and no comparison of values can check a clause stated on bits.

No tolerance fixes both cases. Accepting the first means tolerating a one-step code difference, which a wrong kernel can also produce: a quantizer that truncates instead of rounding passes a one-code tolerance, yet substituted into the $154$ linear layers of TinyLlama-1.1B at INT4 it raises perplexity from 11.4 to 6,648 (Appendix D). The second lies at distance zero, where no tolerance, however strict, rejects it. The problem is not one of tuning: on the three-element input of Section 2, a check that compares codes and scale with the reference's, each on its own, accepts a wrong kernel whenever it accepts a correct one. What decides the first case is whether the codes follow from the scale the kernel returns, which one reconstruction per declared path checks; what decides the second is whether the bits are the ones the contract states.

Each of the 84 contracts of QKERNELBENCH (80 quantized and 4 high-precision baselines) states the numerical semantics of one kernel as a set of clauses and which implementation choices are lawful. Codes returned with a scale are accepted only if some declared path reproduces them bit for bit from an admissible scale, a test we call membership; other codes must match exactly except where the contract lets a code move by one step, and floating-point outputs must lie within declared bounds. Every decision comes with a receipt: a rejection names the failed check and positions, an acceptance the path and scale.

We contribute:

- **Why a quantizer's codes must be judged at the scale it returns.** When the scale may lawfully vary, the same codes can be right under one scale and wrong under another, so any check that compares codes and scale with the reference's separately misjudges some kernel (Section 2). Judging the codes at the scale the kernel returns instead costs one reconstruction per declared arithmetic path.
- **QKERNELBENCH**, 84 contracts for quantized-inference kernels, from INT8, INT4, FP8, FP4 and MX quantizers to low-precision GEMMs, attention caches and fused pipelines. Each contract states its numerical rules clause by clause, and before release its verdicts must reject mutants derived from its own declarations and accept lawful variants of the reference, and every input-conditioned clause the corpus can reach must be triggered by some case (Section 3).
- **An evaluation** of eight LLMs on all 84 contracts, in two rounds, the second generated after the contracts were frozen. Frontier models solve most contracts, but reach only 11% to 27% of the speed of shipping kernels on the 6 contracts where we timed one. On the same kernels, the correctness rules of published benchmarks reject correct kernels and accept wrong ones, and no distance threshold separates the two. After repairs that the shipping

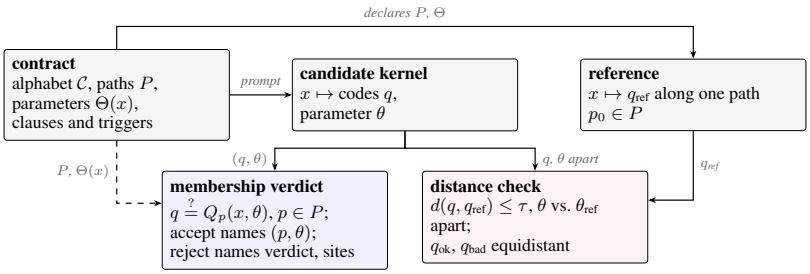



Figure 1: Membership reconstructs the codes from the returned parameter along each declared path and compares bit patterns; a distance check thresholds a function of each output and the reference's, never interpreting the codes by the returned parameter.

kernels of eight libraries prompted, the contracts accept their scale paths, reassociation and tensor-core accumulation; the kernels still rejected follow another convention for a stated clause, compute in a lower precision than the contract admits, or return codes that change between calls (Section 5).

## 2 Why Code Comparisons Cannot Judge a Quantizer

**Setup.** A quantizer maps an input $x \in \mathbb{R}^n$ to a pair $(q, \theta)$. Here $q \in \mathcal{C}^n$ are codes from a finite alphabet $\mathcal{C}$ (for INT8, $\{-127, \dots, 127\}$; for E4M3, the 256 bit patterns), and $\theta$ is the parameter used to produce them, a scale and, for asymmetric schemes, a zero point. Both are returned, because the codes cannot be interpreted without the parameter. A contract fixes a set $\Theta(x)$ of admissible parameters and a set $P$ of admissible arithmetic paths, and declares a candidate that returns $(q, \hat{\theta})$ correct when

$$\hat{\theta} \in \Theta(x) \quad \text{and} \quad \exists\, p \in P :\ q = Q_p(x, \hat{\theta}),$$

where equality means equality of bit patterns. $\Theta(x)$ describes what may vary, such as the last place of a scale, and $P$ how the codes may be computed from the parameter, for example by dividing by the scale or multiplying by its reciprocal. A path may also take its factor from the input rather than from $\hat{\theta}$, as the third path $x \cdot q_{\max}/\mathrm{absmax}$ of vLLM's and SGLang's INT8 quantizers does; $\hat{\theta}$ must then still lie in $\Theta(x)$, and the contract accepts the reference's codes returned with any such $\hat{\theta}$, as those libraries' own outputs are. Figure 1 contrasts this with a distance check.

**Current checks.** KernelBench applies `allclose` to each output (Ouyang et al., 2025); KernelBench-X, which has a dedicated quantization category, thresholds cosine similarity, relative $L_1$ and RMSE jointly (Wang et al., 2026); FlashInfer-Bench thresholds the fraction of elements within one shared `rtol`/`atol` pair (Xing et al., 2026); and a recent design calibrates a per-output tolerance against an `fp64` recomputation (Sarkar, 2026). Applied to a quantizer, each compares the codes with $q_{\text{ref}}$ and a returned scale with $\theta_{\text{ref}}$ separately. None of these rules was written for a quantizer that returns its scale: KernelBench-X's quantization tasks are quantized operators such as a W8A8 matrix product, FlashInfer-Bench's low-bit tasks consume codes, and KernelBench and the calibrated design pose no quantizer, so applying their rules to such codes is our adaptation (Appendix F).

**A construction.** Consider $x = [\,0.5,\ 1.5,\ 127\,]$ under an INT8 symmetric contract with round-half-to-even whose paths divide by the scale or multiply by its reciprocal. The reference computes $\theta_{\text{ref}} = 1$ and $q_{\text{ref}} = [0, 2, 127]$. Let $\theta' = \mathrm{nextafter}(1)$, one binary32 ULP above 1. It lies inside the contract's two-ULP scale budget, which exists because lawful ways of computing a scale can differ by a rounding. Under $\theta'$ the correct codes are $q_{\text{ok}} = [0, 1, 127]$, since $1.5/\theta' < 1.5$ rounds down. A kernel that returns $(q_{\text{ok}}, \theta')$ is correct. A kernel that returns the same codes with $\theta = 1$ is wrong, because at that scale 1.5 rounds to 2. A check that compares the codes with $q_{\text{ref}}$ and the scale with $\theta_{\text{ref}}$, each on its own, sees the same codes in both kernels and the reference's own scale in the wrong one; if it accepts that scale, it accepts the wrong kernel whenever it accepts the correct one. The construction needs $\Theta(x)$ to hold more than one element, as it does in every contract with a scale

budget. With path freedom alone, a check against the codes of every declared path at the reference's scale would suffice, but a distance to the single $q_{\text{ref}}$ does not. The published distances also fail when the codes differ: no $\theta \in \Theta(x)$ produces $q_{\text{bad}} = [0, 3, 127]$ under any path in $P$, yet $q_{\text{ok}}$ and $q_{\text{bad}}$ both satisfy $|q - q_{\text{ref}}| = [0, 1, 0]$. Computed on the codes, $L_1$, $L_\infty$, RMSE, relative $L_1$ and SQNR are identical for the two, `allclose` returns the same verdict, and their cosine similarities differ by $1.5 \times 10^{-8}$, while published cosine thresholds lie between $0.90$ and $0.95$ (Table 7 in Appendix F). Membership separates both pairs, because it reconstructs the codes at the $\theta$ each kernel returns. The FP4 kernel of the introduction is a code error at distance zero.

**Cost of membership.** For a returned $\hat{\theta}$ the admissible codes are $\{Q_p(x, \hat{\theta}) : p \in P\}$, one candidate per path at each position (two where a path's factor is approximate and the site lies within its error of a rounding boundary, other than an exact tie; Appendix B), so a test needs one check that $\hat{\theta} \in \Theta(x)$ and one reconstruction per declared path, with no search over $\Theta(x)$. Floating-point outputs are different. When the summation order is free, the set of lawful results is combinatorially large and the kernel does not return its order, so a bound is the practical check. QKERNELBENCH derives this bound from the operands and the declared accumulation dtype, and uses step bounds or tolerances for other floating-point outputs (Section 3.2).

# 3 THE QKERNELBENCH BENCHMARK

## 3.1 TASKS

The 84 contracts cover the main kernel families of quantized LLM inference, though mixture-of-experts pipelines only in part (Appendix A): INT8 and INT4 quantization and dequantization at four granularities, packing and layout transforms, FP8, MX and NVFP4 formats, INT8 and FP8 matrix multiplication with scaled epilogues, weight-only GEMMs in the style of Marlin, AWQ and GPTQ, attention and KV-cache quantization, dynamic activation quantization, multi-op GEMMs whose dequantized operand never reaches global memory, and fused pipelines. The 4 high-precision baselines repeat quantized shapes at twice the bit width. Shapes come from released model configurations (LLaMA, Qwen3.5, DeepSeek-V4 and Kimi-K3 dimensions, including tensor-parallel splits) and range from decode to prefill, although a serving trace produces none of their power-of-two token counts (Appendix A), which also compares the corpus with the operators SGLang and FlashInfer expose.

We index the corpus by what a quantized kernel can get wrong, besides the four composition tiers of Appendix A. Every clause belongs to one of eight axes (rounding, saturation, parameter, degenerate, storage, accumulation, composition and structure; Table 3 in Appendix A), which a release gate enforces.

## 3.2 CONTRACTS AND VERDICTS

Each contract ships a machine-readable declaration, its cases, a prose specification and a PyTorch reference. The declaration lists the numerics clauses and, for each, the input condition under which the clause becomes observable, drawn from a closed vocabulary (`tie_sites`, `at_clamp`, `zero_rows`, `non_finite`, ...). Random inputs rarely hit a tie or a clamp, so each contract adds constructed boundary cases, and a gate checks that each of the 89 input-conditioned clauses is triggered by some case; 15 more, reachable only through a transform the corpus does not construct, are declared as such.

The declaration also gives every output a verdict class according to its type. A code output whose parameter the kernel returns is decided by *membership*. For such an output the declaration specifies the admissible parameters as a budget in representable steps or, for two transforms, a relative bound, the admissible paths (`divide`, `reciprocal`, the *third path* $x \cdot q_{\max}/\text{absmax}$ of vLLM's and SGLang's INT8 quantizers, or a named transform) and the positions where no freedom is granted. Other code outputs are compared exactly after unpacking, except at positions where a declared freedom lets a code move by one step; a mask computed from the contract and the input marks these positions (Appendix B). A floating-point output whose operation order the contract fixes must lie within $k$ representable steps of the reference. One whose summation order is free must satisfy an order-independent forward-error bound computed from the operands and the declared accumulation

dtype where we derived one (Appendix C); the other dense outputs carry a value tolerance set by hand or from measurement, 26 of the 109 declared outputs; ten of them exceed five per cent of the output's median magnitude, the loosest 3.4 times it, and the kernels these contracts accept on their own case use at most 44% of their bound but for one `w8a8_linear` kernel (Appendix J). A gate rejects any value tolerance on a code output. Top-$k$ selection and which entries a paged cache write touches are decided by structural relations. Every decision produces a receipt. A rejection names the failed verdict, the number of violating positions and the first of them, and a membership acceptance names the path and parameter that reproduce the codes.

### 3.3 Calibration and release gates

A declaration is useful only if the verdict can observe what it declares. For each contract, a calibration step generates mutants from the declaration: defects that break its clauses (ties broken the other way, a scale one step outside the budget, saturation removed, a narrower accumulator) and lawful variants for its declared freedoms (the other path, a scale inside the budget, split-K, a wider accumulator). The verdict must reject every defect and accept every lawful variant. A clause that no mutant can expose (inert) or a verdict that rejects a lawful variant (over-strict) blocks the release. All 84 contracts pass against 318 mutants, 225 defects and 93 lawful variants. Calibration shows that each clause has a defect its verdict rejects. Whether a verdict admits a lawful choice the contract does not declare, such as an undeclared reduction order or fused multiply-adds (four contracts repaired after judging had rejected one of these; Appendix K.3) or tensor-core accumulation, is tested with shipping kernels (Section 5.3), and the campaign version's verdict code with an independent checker (Section 5.4). Calibration and the checks on the reference, which is judged under the lawful alternatives that apply to it (split-K, wider accumulation and fused multiply-adds in a reduction, the other scale path, another device), are among the 30 release gates (Table 5), all passing on the version we will release.

### 3.4 Harness and scoring

Each candidate is compiled and run in a fresh process, in an environment that holds no credentials, on every declared case at the visible seed, the one the prompt's input generator shows, and at two held-out seeds that appear nowhere in the prompt. The contract's *own case* is the first workload case it declares. A kernel *solves* a contract only if the contract accepts it on every declared case at every seed. Speed is the ratio of the reference's run time to the kernel's on the contract's own case, measured with CUDA events in a fresh process for solved kernels and aggregated as a geometric mean.

## 4 Experimental Setup

**Models and generation.** We evaluate eight models: six frontier systems from six developers (Claude Sonnet 5, GPT-5.6-sol, Kimi-K3, DeepSeek-V4-Pro, GLM-5.3 and Qwen3.8-Max) and two open-weight models of one family (Qwen3.5-397B-A17B and Qwen3.5-35B-A3B). All models are accessed through one API gateway with the same client, which sends a single request without tools, and each model is asked once per contract in each of two rounds, the second after the contracts were frozen (Section 5.4). The prompt contains the contract, not the reference, and describes the build environment (Appendix E). We request no extended-thinking budget; the output budget is 32,000 tokens, and calls that spent it all on reasoning without code (45 in the first round, 52 in the second) were repeated once with 120,000 tokens.

**Judging.** Every kernel is judged on an RTX PRO 6000 Blackwell (`sm_120`), on each declared case of its contract at the visible seed and at two held-out seeds, 7,554 cells (one case at one seed) over the 530 first-round kernels that build, pooled over the eight models, against the *repaired release*, the corpus after the changes of Section 5.4, whose verdicts carry over to the released corpus (Table 13). In every cell the contract and five published rule settings judge the same outputs. A shared interface layer reconciles the rank of each output and turns away, for every rule, a wrong number of outputs or an output of the wrong bit width or size; INT4 codes are compared as packed words. The settings are KernelBench's `allclose` with $\texttt{rtol} = \texttt{atol} = 10^{-2}$; KernelBench-X's joint cosine, relative-$L_1$ and RMSE thresholds, at the strict end of their published ranges; FlashInfer-Bench's matched-ratio

Table 1: The first round, per model, one request on each of the 84 contracts. Built, accepted, rejected and solved are judged on the RTX PRO 6000 (definitions in Section 5.1); speed is timed on the RTX 6000 Ada. The second round is in Table 15.

| model | built | accepted | rejected | solved | speed | speed on shared |
|---|---|---|---|---|---|---|
| Claude Sonnet 5 | 80 | 76 | 4 | 72 | 3.6× | 4.1× |
| GPT-5.6-sol | 79 | 78 | 1 | 77 | 5.0× | 4.8× |
| Kimi-K3 | 82 | 80 | 2 | 80 | 4.2× | 4.7× |
| DeepSeek-V4-Pro | 71 | 71 | 0 | 71 | 3.6× | 3.9× |
| GLM-5.3 | 80 | 79 | 1 | 79 | 4.6× | 4.8× |
| Qwen3.8-Max | 80 | 79 | 1 | 79 | 3.9× | 4.5× |
| Qwen3.5-397B-A17B | 52 | 40 | 12 | 39 | 3.0× | — |
| Qwen3.5-35B-A3B | 6 | 4 | 2 | 4 | 4.7× | — |

rule, at the ratio its configuration resolves to by default (1.0) and at the 0.95 reported in its paper; and a per-output tolerance calibrated against an `fp64` recomputation of the reference, following Sarkar (2026). KernelBench's rule and the calibrated tolerance were not written for codes, and Appendix F specifies how we apply each rule.

**The development population.** Before this campaign, 1,032 attempts by eight models over three rounds on 43 contracts (the *development campaign*) and nine single-shot runs on the other 41 (the *archived run*) used an earlier prompt; we developed the contracts on them and do not rank models on them (Appendix L).

# 5 Results

## 5.1 What the models achieve

Table 1 reports three stages for each model: a kernel is *built* if it compiles and runs on its contract's own case (the *decidable* kernels), *accepted* if the contract accepts it there at the visible seed, and *solved* if it is accepted on every declared case at three seeds on the RTX PRO 6000. *Speed*, timed on the RTX 6000 Ada, is over each model's solved kernels that the contract also accepts on their Ada timing inputs (500 of the 501 first-round solved kernels), and *speed on shared* over the 49 contracts on which every frontier model has one. The prompts state the clauses that the verdicts check, with the exceptions of Appendix J, so the solved counts measure whether a model implements a stated specification; they are not comparable with pass rates under less complete task statements, such as those of Section 1. Per model, solved kernels run 3.0× to 5.0× faster than the contract's portable PyTorch reference, which `torch.compile` speeds up by 1.18× but, compiled, fails 10 contracts (Appendix E). Against the shipping kernel on its own case, each frontier model's solved kernels reach 11% to 27% of its speed on the 6 contracts where one runs on Ada, four of them GEMMs (Appendix E).

**Compilation.** Each frontier model returns a kernel that builds and runs on 71 to 82 of the 84 contracts, and the two Qwen3.5 models on 52 and 6 (Table 1). Of those kernels, the contract rejects 0 to 4 of each frontier model's, 12 of Qwen3.5-397B-A17B's and 2 of Qwen3.5-35B-A3B's (Appendix E classifies the build failures). Of the 507 first-round kernels accepted on their own case at the visible seed, six fail another case or seed, each by breaking a clause its prompt states.

**Rejections.** Of the 23 first-round kernels that the contract rejects on their own case, 4 return codes that no declared path reproduces at the scale they return, 7 exceed a declared bound on a floating-point output, 6 differ from an output that must be exact, among them the FP4 decoder, and 6 fail the shared interface layer.

Table 2: Disagreement with the contract on each kernel's own case at the visible seed, over the 530 first-round kernels that build (evaluation) and the 716 decidable kernels of the development population. The 6 and 13 kernels that return raw storage bytes, which the contract accepts and every value rule rejects, are counted in neither column. The calibrated tolerance's wrongful rejections come from two choices of our adaptation, a `float32` floor and INT4 codes compared as packed words; without them it rejects none and accepts 4 (Appendix F).

| | | evaluation | | development | |
|---|---|---|---|---|---|
| rule | what it asks of an output | accepts wrongly | rejects wrongly | accepts wrongly | rejects wrongly |
| FlashInfer-Bench, ratio 0.95 | 95% of elements in tolerance | 6 | 6 | 21 | 2 |
| KernelBench-X | cosine, $L_1$ and RMSE | 6 | 0 | 16 | 0 |
| calibrated tolerance | inside the `fp64` envelope | 3 | 70 | 7 | 79 |
| KernelBench `allclose` | every element in tolerance | 2 | 58 | 2 | 44 |
| FlashInfer-Bench, ratio 1.0 | every element in tolerance | 2 | 59 | 2 | 44 |

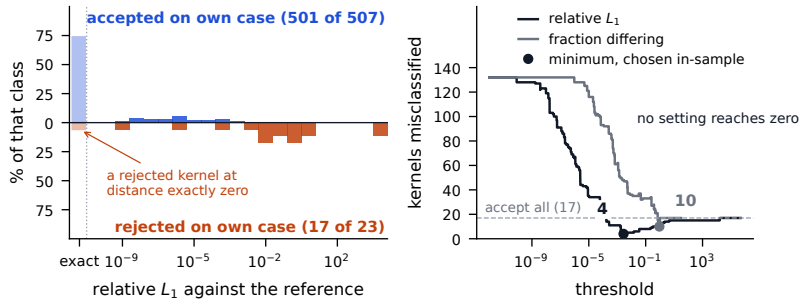



Figure 2: No threshold on either distance separates the kernels the contract accepts from those it rejects. *Left*: the 530 first-round kernels that build, less the 6 storage-byte kernels and the 6 interface failures (518), by the relative $L_1$ of their first output; each class is normalized to itself, and distance zero has its own bin. *Right*: kernels misclassified at every threshold on either distance; the dashed line is the 17 errors of accepting every kernel.

### 5.2 The published rules on the same kernels

Table 2 compares each rule setting with the contract on the same outputs: on the 530 first-round kernels that compile and run, pooled over the eight models (the built column of Table 1) and on the 716 decidable kernels of the development population. Every count is a number of kernels. A rule *accepts wrongly* when it accepts a kernel that the contract rejects, and *rejects wrongly* when it rejects a kernel that the contract accepts.

**Both kinds of error.** The strict settings, `allclose` and FlashInfer-Bench at its default ratio, err mostly by not reading the returned parameter: 47 of `allclose`'s 58 wrongful rejections are on the 13 contracts that judge exactly one output by membership, all from kernels whose returned scale differs from the reference's, the situation of Section 2 (Appendix F); the calibrated tolerance's wrongful rejections come from our adaptation (Table 2). The loose settings, KernelBench-X and FlashInfer-Bench at 0.95, accept 6 and 6 kernels that violate a declared clause. Each, like the FP4 decoder, is wrong at few positions or by little at each, which an aggregate over the tensor does not detect (Appendix F lists them). Not all of these errors involve a returned scale: 4 of KernelBench-X's 6 wrongful acceptances and 11 of `allclose`'s 58 wrongful rejections are on contracts that judge no output by membership. Per model over every case (Table 9), `allclose` credits each frontier model with 14 to 18 fewer contracts than the contract in both rounds, more than the frontier models differ; the loose settings stay within 3, and FlashInfer-Bench at 0.95 credits Claude Sonnet 5 with 3 more by accepting kernels the contract rejects. Since no change turned an acceptance into a rejection (Appendix K.6), 28 of `allclose`'s 58 wrongful rejections and all 6 of KernelBench-X's wrongful acceptances hold under every version of the corpus that judged them.

**Thresholds.** Figure 2 places the decidable kernels by their distance from the reference. Wrong kernels occur at every distance down to zero, where the FP4 decoder sits, so a threshold loose enough to accept every correct kernel also accepts some wrong ones: the best single threshold on relative $L_1$, chosen with the verdicts in hand, accepts 4 of the 17 wrong kernels, and the best on the fraction of differing elements accepts 10. A separate threshold for each contract, fitted to the verdicts it is scored against, still misclassifies 2, because 2 of the 13 contracts with kernels in both classes are inseparable.

### 5.3 SHIPPING KERNELS

A contract can be consistent and still describe a convention no implementation uses, so we ran every shipping kernel of SGLang, FlashInfer, cuBLAS, PyTorch, vLLM, torchao, DeepGEMM and TensorRT-LLM that we could run and that computes a contract's operation on an RTX 6000 Ada, the RTX PRO 6000 or a GB10, unmodified, on every case at three seeds (Appendix G). These kernels shaped the repairs, so they are not a held-out test of them: the repaired contracts admit the tensor-core group accumulation of the FP8 GEMMs, the half-precision probabilities of the decode kernels, the third scale path of the INT8 quantizers and torchao's reading of a one-sided zero-point range (Appendices C and K.4). Under the released corpus, 28 of the 34 pairs of a library kernel and a contract from the first four libraries and 40 of the 62 from the other four pass every case they run, DeepGEMM only on inputs whose scales are powers of two. The others differ from a clause their contract states or, for the MX quantizers of FlashInfer 0.6.1 and TensorRT-LLM, return codes on Ada that change from call to call; for torchao's affine quantizer on seven contracts and the GPTQ GEMM of SGLang and vLLM, that clause, the precision of a division or an accumulation, was stated only after these rejections and no verdict reads it (Appendix K.5).

### 5.4 VALIDATING THE INSTRUMENT

**Changes to the contracts.** We changed 21 contracts after judging the evaluation's kernels, and 16 of them had rejected a correct output we recorded (Table 12 in Appendix K). Of the 527 first-round kernels decided under both versions, the contracts rejected 52 before these changes that they now accept, 30 on their own case, among them all but 5 of the 30 that KernelBench-X then appeared to accept wrongly. Receipts of both populations exposed five: four INT4 quantizers required exact codes while their scale verdict admitted two ULPs, the incoherence of Section 2 in our own corpus, and `llm_int8_linear`'s reference divided its INT8 path by an extra $127^2$. Reading every kernel that the campaign version rejected found seven more contracts that rejected 22 correct kernels, mostly over free choices such as a reduction's order, and the shipping kernels found four more (Section 5.3). No change turned a recorded kernel from accepted to rejected, and the 65 contracts the repaired release leaves unchanged reproduce all 5,676 of their cell verdicts; two more change only in the released corpus, by a statement and an alternative reference that decide no recorded verdict.

**A held-out round.** Because these changes followed the kernels, we froze the contracts, asked every model once more and judged the new kernels with nothing changed (Appendix N). The round can find a contract that rejects a correct kernel, not one that accepts a wrong kernel. Per model, the solved count moves by at most 7. Of the 35 second-round kernels that build but are not solved, 33 break a clause their prompt states (2 return FP32 where it states FP16), so the frozen contracts reject none of them wrongly, and 2 miss the tolerance almost everywhere on the two contracts whose FP8 names contradict their INT8 data, which we count neither way. Most kernels repeat a first-round output; on the 91 that do not, `allclose` rejects 11 correct ones and KernelBench-X accepts 6 wrong ones.

**Independent checks.** An independent checker of the campaign version's decision procedure, sharing no code with our verdict layer, and exact models of membership and of MX outputs read every output of the 586 decidable development-campaign kernels and of 504 first-round kernels (packed INT4 codes excepted); no disagreement is arithmetic (Appendix H). These checks test our code, not the repairs of Table 12. On Ada (`sm_89`) and Blackwell (`sm_120`) with identical inputs, the verdicts on the 716 decidable development kernels differ in 6 of 9,316 cells, none about a code the contract requires exactly; the evaluation's own-case verdicts are the same on the Blackwell card and a GB10

(`sm_121`), which solves one fewer (Appendix I), and one kernel solved on the RTX PRO 6000 fails every cell on Ada (Appendix E).

## 6 Related Work

**Kernel-generation benchmarks.** KernelBench framed the task as translating PyTorch modules into fast CUDA (Ouyang et al., 2025); TritonBench, MultiKernelBench and KernelGenBench extended it to Triton, to other platforms and to per-operator token cost (Li et al., 2025a; Wen et al., 2025; FlagOS-AI, 2026). Later work evaluated agent workflows and production stacks (Younesian et al., 2026; Oliaro et al., 2026), selected tasks from serving traces (Xing et al., 2026; Yang et al., 2026), scored kernels against speed-of-light bounds (Lin et al., 2026; Huang et al., 2026; Hari et al., 2026), and hardened evaluation with hidden input distributions, fuzzing and replacement audits (Zhang et al., 2026; Sarkar, 2026; Li et al., 2026). These benchmarks treat low-precision dtypes as workloads with fixed reference semantics; in QKernelBench, the quantization semantics are what the task specifies and the verdict checks (Table 4, Appendix A).

**Auditing the correctness signal.** Sarkar (2026) shows that hand-picked thresholds are orders of magnitude looser than a kernel's measured error and calibrates per-output tolerances against `fp64`. Shah & Shrestha (2026) audit 2,638 machine-generated kernels with twelve adversarial gates and find the standard test far too loose for dense kernels. KernelBench-X (Wang et al., 2026) is the closest benchmark: its six quantized operators are judged by joint cosine, relative-$L_1$ and RMSE thresholds. Kernel Contracts (Veit, 2026) declare precision and order semantics and calibrate against good, bad and baseline kernels; our contracts add membership at the returned parameter, a trigger and a constructed case for each input-conditioned clause, and mutants derived from each clause, a form of mutation testing (DeMillo et al., 1978). For quantized outputs the standard test errs in both directions, and none of these works checks whether a returned parameter explains the returned codes.

**Kernel generation and quantization systems.** Reinforcement learning and agent loops generate kernels (Baronio et al., 2025; Li et al., 2025c;b; Wang et al., 2025; Lange et al., 2025a), and the contracts draw on the quantization literature (Dettmers et al., 2022; Frantar et al., 2022; Xiao et al., 2023; Lin et al., 2024; Ashkboos et al., 2024; Zhao et al., 2024; Lin et al., 2025) and on production kernels and serving libraries (Frantar et al., 2025; DeepSeek-AI, 2025; Shah et al., 2024; NVIDIA Corporation, 2025; Zheng et al., 2024; Kwon et al., 2023; Ye et al., 2025; NVIDIA, 2023; PyTorch Team, 2024).

## 7 Limitations and Conclusion

**Limitations.** A contract admits only the conventions it declares, so a kernel that follows another lawful convention is rejected, and the contracts that no shipping kernel exercises rest on their references, mutants and generated kernels. Some contracts were revised after the first round; a second round, generated after the last change, repeats both kinds of rule error and none of its rejected kernels is correct, except possibly two on the contracts whose FP8 names contradict their INT8 data (Section 5.4). The comparison with published rules depends on how rules written for dense outputs are applied to codes (Appendix F), and single-shot rounds measure conformance to stated numerics rather than rank models.

**Conclusion.** QKernelBench judges quantized codes by membership at the returned parameter and attaches a receipt to every decision. On it, frontier models meet stated quantization numerics on most contracts but, where we timed shipping kernels against them, reach a fraction of their speed, while the published rules we evaluate reject correct kernels whose scale differs lawfully and accept kernels that break a stated clause. We suggest that benchmarks with quantized outputs judge codes at the parameter the kernel returns.

### Reproducibility Statement

Section 3 and Appendices A–C specify the contracts and their verdicts, Section 4 and Appendix E the evaluation (models, prompts, budgets, devices and seeds), and Appendix F how each published rule is applied. Every generated kernel is compiled and run in a fresh process, in an environment

with no credentials, on every declared case at the visible seed and two held-out seeds. The corpus carries a content digest, every judging record stores the digest, device and framework build it was measured with, and Appendix K lists every version of the corpus and every change to it. Inputs come from deterministic generators, but the same seed draws different tensors on different GPUs and framework builds (Appendix I), so a verdict reproduces exactly on the recorded device class and build; timings must be measured again on other devices. Upon acceptance we will release under an open license the corpus, the input tensors of every case, the verdict layer, the 30 release gates, every generated kernel, every judging record, the analysis tools and a script that recomputes every number in this paper from those records.

### Ethics Statement

This work evaluates generated GPU kernels against numerical specifications. It involves no human subjects, no personal data, and no deployment of the generated code. Shipping kernels of eight libraries are run unmodified and compared against our contracts; where they differ, we report the clause they differ in, or the behavior we observed, and do not characterize any library as defective.

### AI Use Statement

LLM-based tools assisted with coding, experiments and writing. The authors reviewed all AI-assisted work and take full responsibility for the content of this paper.

## References

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. QuaRot: Outlier-free 4-bit inference in rotated LLMs. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/b5b939436789f76f08b9d0da5e81af7c-Abstract-Conference.html.

Carlo Baronio, Pietro Marsella, Ben Pan, Simon Guo, and Silas Alberti. Kevin: Multi-turn RL for generating CUDA kernels. *CoRR*, abs/2507.11948, 2025. doi: 10.48550/ARXIV.2507.11948. URL https://doi.org/10.48550/arXiv.2507.11948.

DeepSeek-AI. DeepSeek-V3 technical report. *CoRR*, abs/2412.19437, 2024. doi: 10.48550/ARXIV.2412.19437. URL https://doi.org/10.48550/arXiv.2412.19437.

DeepSeek-AI. DeepGEMM: Clean and efficient FP8 GEMM kernels with fine-grained scaling. https://github.com/deepseek-ai/DeepGEMM, 2025.

Richard A. DeMillo, Richard J. Lipton, and Frederick G. Sayward. Hints on test data selection: Help for the practicing programmer. *Computer*, 11(4):34–41, 1978.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. LLM.int8(): 8-bit matrix multiplication for transformers at scale. *CoRR*, abs/2208.07339, 2022. doi: 10.48550/ARXIV.2208.07339. URL https://doi.org/10.48550/arXiv.2208.07339.

FlagOS-AI. KernelGenBench: A multi-source and multi-chip benchmark for LLM-based kernel generation. GitHub manuscript, https://github.com/flagos-ai/KernelGenBench, 2026.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: accurate post-training quantization for generative pre-trained transformers. *CoRR*, abs/2210.17323, 2022. doi: 10.48550/ARXIV.2210.17323. URL https://doi.org/10.48550/arXiv.2210.17323.

Elias Frantar, Roberto L. Castro, Jiale Chen, Torsten Hoefler, and Dan Alistarh. MARLIN: mixed-precision auto-regressive parallel inference on large language models. In *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming, PPoPP 2025, Las Vegas, NV, USA, March 1-5, 2025*, pp. 239–251. ACM, 2025. doi: 10.1145/3710848.3710871. URL https://doi.org/10.1145/3710848.3710871.

Siva Kumar Sastry Hari, Vignesh Balaji, Sana Damani, Qijing Huang, and Christos Kozyrakis. Improving efficiency of GPU kernel optimization agents using a domain-specific language and speed-of-light guidance. *CoRR*, abs/2603.29010, 2026. doi: 10.48550/ARXIV.2603.29010. URL https://doi.org/10.48550/arXiv.2603.29010.

Nicholas J. Higham. *Accuracy and stability of numerical algorithms, Second Edition*. SIAM, 2002. ISBN 978-0-89871-521-7. doi: 10.1137/1.9780898718027. URL https://doi.org/10.1137/1.9780898718027.

Qijing Huang, Sana Damani, Zhifan Ye, Athinagoras Skiadopoulos, Siva Kumar Sastry Hari, Jason Clemons, Sahil Modi, Jingquan Wang, Aditya Kane, Edward Lin, Humphrey Shi, and Christos Kozyrakis. SOLAR: AI-powered speed-of-light performance analysis. *CoRR*, abs/2606.26383, 2026. doi: 10.48550/ARXIV.2606.26383. URL https://doi.org/10.48550/arXiv.2606.26383.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023.

Robert Tjarko Lange, Aaditya Prasad, Qi Sun, Maxence Faldor, Yujin Tang, and David Ha. The AI CUDA engineer: Agentic CUDA kernel discovery, optimization and composition. Sakana AI Technical Report, https://pub.sakana.ai/ai-cuda-engineer, 2025a.

Robert Tjarko Lange, Qi Sun, Aaditya Prasad, Maxence Faldor, Yujin Tang, and David Ha. Towards robust agentic CUDA kernel benchmarking, verification, and optimization. *CoRR*, abs/2509.14279, 2025b. doi: 10.48550/ARXIV.2509.14279. URL https://doi.org/10.48550/arXiv.2509.14279.

Jianling Li, Shangzhan Li, Zhenye Gao, Qi Shi, Yuxuan Li, Zefan Wang, Jiacheng Huang, Haojie Wang, Jianrong Wang, Xu Han, Zhiyuan Liu, and Maosong Sun. TritonBench: Benchmarking large language model capabilities for generating Triton operators. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, volume ACL 2025 of *Findings of ACL*, pp. 23053–23066. Association for Computational Linguistics, 2025a. doi: 10.18653/V1/2025.FINDINGS-ACL.1183. URL https://doi.org/10.18653/v1/2025.findings-acl.1183.

Shangzhan Li, Zefan Wang, Ye He, Yuxuan Li, Qi Shi, Jianling Li, Yonggang Hu, Wanxiang Che, Xu Han, Zhiyuan Liu, and Maosong Sun. AutoTriton: Automatic Triton programming with reinforcement learning in LLMs. *CoRR*, abs/2507.05687, 2025b. doi: 10.48550/ARXIV.2507.05687. URL https://doi.org/10.48550/arXiv.2507.05687.

Tingxi Li, Ravishka Rathnasuriya, and Wei Yang. Correct but slow: An empirical study of the GPU kernel evaluation gap in modern domain-specific languages. *CoRR*, abs/2607.04454, 2026. doi: 10.48550/ARXIV.2607.04454. URL https://doi.org/10.48550/arXiv.2607.04454.

Xiaoya Li, Xiaofei Sun, Albert Wang, Jiwei Li, and Chris Shum. CUDA-L1: improving CUDA optimization via contrastive reinforcement learning. *CoRR*, abs/2507.14111, 2025c. doi: 10.48550/ARXIV.2507.14111. URL https://doi.org/10.48550/arXiv.2507.14111.

Edward Lin, Sahil Modi, Siva Kumar Sastry Hari, Qijing Huang, Zhifan Ye, Nestor Qin, Fengzhe Zhou, Yuan Zhang, Jingquan Wang, Sana Damani, Dheeraj Peri, Ouye Xie, Aditya Kane, Moshe Maor, Michael Behar, Triston Cao, Rishabh Mehta, Vartika Singh, Vikram Sharma Mailthody, Terry Chen, Zihao Ye, Hanfeng Chen, Tianqi Chen, Vinod Grover, Wei Chen, Wei Liu, Eric Chung, Luis Ceze, Roger A. Bringmann, Cyril Zeller, Michael Lightstone, Christos Kozyrakis, and Humphrey Shi. SOL-ExecBench: Speed-of-light benchmarking for real-world GPU kernels against hardware limits. *CoRR*, abs/2603.19173, 2026. doi: 10.48550/ARXIV.2603.19173. URL https://doi.org/10.48550/arXiv.2603.19173.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: activation-aware weight quantization for on-device LLM compression and acceleration. In Phillip B. Gibbons, Gennady Pekhimenko, and

Christopher De Sa (eds.), *Proceedings of the Seventh Annual Conference on Machine Learning and Systems, MLSys 2024, Santa Clara, CA, USA, May 13-16, 2024*. mlsys.org, 2024.

Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. QServe: W4A8KV4 quantization and system co-design for efficient LLM serving. In Matei Zaharia, Gauri Joshi, and Yingyan (Celine) Lin (eds.), *Proceedings of the Eighth Conference on Machine Learning and Systems, MLSys 2025, Santa Clara, CA, USA, May 12-15, 2025*. OpenReview.net/mlsys.org, 2025. URL https://openreview.net/forum?id=1FfmStySS1.

Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, Naveen Mellempudi, Stuart F. Oberman, Mohammad Shoeybi, Michael Y. Siu, and Hao Wu. FP8 formats for deep learning. *CoRR*, abs/2209.05433, 2022. doi: 10.48550/ARXIV.2209.05433. URL https://doi.org/10.48550/arXiv.2209.05433.

NVIDIA. TensorRT-LLM. https://github.com/NVIDIA/TensorRT-LLM, 2023.

NVIDIA Corporation. CUTLASS: CUDA templates for linear algebra subroutines. https://github.com/NVIDIA/cutlass, 2025.

Gabriele Oliaro, Yichao Fu, May Jiang, Owen Lu, Junli Wang, Zhihao Jia, Hao Zhang, and Samyam Rajbhandari. FastKernels: Benchmarking GPU kernel generation in production. *CoRR*, abs/2605.23215, 2026. doi: 10.48550/ARXIV.2605.23215. URL https://doi.org/10.48550/arXiv.2605.23215.

Open Compute Project. OCP microscaling formats (MX) specification, version 1.0, 2023.

Anne Ouyang, Simon Guo, Simran Arora, Alex L. Zhang, William Hu, Christopher Ré, and Azalia Mirhoseini. KernelBench: Can LLMs write efficient GPU kernels? In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/ouyang25a.html.

PyTorch Team. torchao: PyTorch-native quantization and sparsity. https://github.com/pytorch/ao, 2024.

Dipankar Sarkar. The correctness illusion in LLM-generated GPU kernels. *CoRR*, abs/2606.20128, 2026. doi: 10.48550/ARXIV.2606.20128. URL https://doi.org/10.48550/arXiv.2606.20128.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. FlashAttention-3: Fast and accurate attention with asynchrony and low-precision. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/7ede97c3e082c6df10a8d6103a2eebd2-Abstract-Conference.html.

Rishi Shah and Rishav Shrestha. A contract-grade verifier for LLM-generated GPU kernels, and a native Blackwell backward for the gated-linear-recurrence family. *CoRR*, abs/2608.12700, 2026. doi: 10.48550/ARXIV.2608.12700. URL https://doi.org/10.48550/arXiv.2608.12700.

Cooper Veit. Kernel contracts: A specification language for ML kernel correctness across heterogeneous silicon. *CoRR*, abs/2604.22032, 2026. doi: 10.48550/ARXIV.2604.22032. URL https://doi.org/10.48550/arXiv.2604.22032.

Han Wang, Jintao Zhang, Kai Jiang, Haoxu Wang, Jianfei Chen, and Jun Zhu. KernelBenchX: A comprehensive benchmark for evaluating LLM-generated GPU kernels. *CoRR*, abs/2605.04956, 2026. doi: 10.48550/ARXIV.2605.04956. URL https://doi.org/10.48550/arXiv.2605.04956.

Jianghui Wang, Vinay Joshi, Saptarshi Majumder, Xu Chao, Bin Ding, Ziqiong Liu, Pratik Prabhanjan Brahma, Dong Li, Zicheng Liu, and Emad Barsoum. GEAK: Introducing Triton kernel AI agent & evaluation benchmarks. *CoRR*, abs/2507.23194, 2025. doi: 10.48550/ARXIV.2507.23194. URL https://doi.org/10.48550/arXiv.2507.23194.

Zhongzhen Wen, Yinghui Zhang, Zhong Li, Zhongxin Liu, Linna Xie, and Tian Zhang. MultiKernelBench: A multi-platform benchmark for kernel generation. *CoRR*, abs/2507.17773, 2025. doi: 10.48550/ARXIV.2507.17773. URL https://doi.org/10.48550/arXiv.2507.17773.

Guangxuan Xiao, Ji Lin, Mickaël Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and efficient post-training quantization for large language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 38087–38099. PMLR, 2023. URL https://proceedings.mlr.press/v202/xiao23c.html.

Shanli Xing, Yiyan Zhai, Alexander Jiang, Yixin Dong, Yong Wu, Zihao Ye, Charlie Ruan, Yingyi Huang, Yineng Zhang, Liangsheng Yin, Aksara Bayyapu, Luis Ceze, and Tianqi Chen. FlashInfer-Bench: Building the virtuous cycle for AI-driven LLM systems. *CoRR*, abs/2601.00227, 2026. doi: 10.48550/ARXIV.2601.00227. URL https://doi.org/10.48550/arXiv.2601.00227.

Lingyun Yang, Yuxiao Wang, Shenghao Liang, Linfeng Yang, Daocheng Ying, Chunbo You, Rui Zhang, Luping Wang, Yinghao Yu, Guodong Yang, and Liping Zhang. Are LLM-generated GPU kernels production-ready? A trace-driven benchmark and optimization agent. *CoRR*, abs/2607.14541, 2026. doi: 10.48550/ARXIV.2607.14541. URL https://doi.org/10.48550/arXiv.2607.14541.

Zihao Ye, Lequn Chen, Ruihang Lai, et al. FlashInfer: Efficient and customizable attention engine for LLM inference serving. In *Proceedings of Machine Learning and Systems*, 2025.

Sharareh Younesian, Wenwen Ouyang, Sina Rafati, Mehdi Rezagholizadeh, S. Kevin Zhou, Ji Liu, Yue Liu, Yuchen Yang, Hao Li, Ziqiong Liu, Dong Li, Vikram Appia, Zhenyu Gu, and Emad Barsoum. AgentKernelArena: Generalization-aware benchmarking of GPU kernel optimization agents. *CoRR*, abs/2605.16819, 2026. doi: 10.48550/ARXIV.2605.16819. URL https://doi.org/10.48550/arXiv.2605.16819.

Yunxiang Zhang, Ping Yu, Jianyu Wang, Max Fan, Julian Reed, Azalia Mirhoseini, and Will Su. KernelBench-Verified: Do LLM-generated kernels actually beat PyTorch? *CoRR*, abs/2607.16241, 2026. doi: 10.48550/ARXIV.2607.16241. URL https://doi.org/10.48550/arXiv.2607.16241.

Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. Atom: Low-bit quantization for efficient and accurate LLM serving. In Phillip B. Gibbons, Gennady Pekhimenko, and Christopher De Sa (eds.), *Proceedings of the Seventh Annual Conference on Machine Learning and Systems, MLSys 2024, Santa Clara, CA, USA, May 13-16, 2024*. mlsys.org, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/5edb57c05c81d04beb716ef1d542fe9e-Abstract-Conference.html.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, 2024.

## CONTENTS OF THE APPENDIX

Appendices A to J follow the order of the paper's sections. Appendix A documents the corpus and compares it with other benchmarks, Appendix B describes contracts and verdict classes, and Appendix C derives the order-independent bound for reductions. Appendix D covers calibration and an end-to-end check. Appendix E details the evaluation campaign, Appendix F the published rules and their comparison with the contracts, and Appendix G the shipping kernels. Appendices H, I and J check the instrument: an independent implementation, other devices, and how much each verdict admits. Appendix K lists every change made to the corpus after the evaluation was judged, and every version of the corpus. Appendix L describes the development population, Appendix M what is not finished, and Appendix N a second round asked after the contracts were frozen. The

code we will release upon acceptance recomputes every measured number in this paper from the records it comes from.

**Conventions.** Three GPUs appear. The *Ada card* is an RTX 6000 Ada (`sm_89`), which times the kernels of Table 1 and judges them again on the inputs they were timed with; the *RTX PRO 6000* Blackwell (`sm_120`, "Blackwell" in tables) judges them; the *GB10* (`sm_121`) repeats the judging and the timing (Appendix I). The four versions of the corpus (the one the prompts were rendered from, the *campaign version*, the *repaired release* and the *released corpus*, the version we will release) are defined in Appendix K.1. The *evaluation campaign* is the first round of one request per model and contract, and Appendix N reports a second round asked after the contracts were frozen; the *development population* is the earlier kernels of Appendix L. A *cell* is one case at one seed, a kernel's *own case* is the first workload case of its contract, a kernel is *decidable* if it builds and runs on its own case, and it is *solved* if its contract accepts it on every cell. A rule *accepts wrongly* when it accepts a kernel that the contract rejects, and *rejects wrongly* when it rejects a kernel that the contract accepts. The *adjudicator* is our implementation of the verdicts.

# A THE CORPUS

**Duplicates.** After comments, docstrings and whitespace are stripped, the 84 contracts have 84 distinct references. 8 of them are granularity variants of an operation that another contract covers at a production granularity, and 8 come from a CNN-era annex that predates the LLM-serving corpus.

**Tiers.** A contract is a kernel, not an algorithm, so that calling a library is not mistaken for writing a kernel; `--use_fast_math` is disallowed, and shapes are real model dimensions with tensor-parallel splits. The four tiers run from data formats (quantize and dequantize over INT8 and INT4 at four granularities, packing, and unit contracts that each isolate one clause) through low-precision GEMMs and multi-op GEMMs whose dequantized operand never reaches global memory to fused pipelines such as LLM.int8() decomposition, AWQ and GPTQ linear layers, FP8 KV-cache decode attention and MLA latent-cache quantization.

**Coverage of what the field ships.** We placed by hand every quantization operator that SGLang and FlashInfer expose. Of 123 names, 39 are not kernels (configurations, runners, JIT generators, layout helpers without arithmetic), leaving 84 operators. A contract implements 18 outright; 47 are fused or grouped forms whose constituents the corpus covers but whose fusion it does not test; and 19 are absent. Operators that share a contract's name but compute another operation (FlashInfer's `layernorm_quant` and its RoPE quantizers take a static FP8 scale where our contracts return one, and `packbits` packs booleans) are not counted as implemented. Ten of the nineteen are mixture-of-experts pipelines and four are quantized attention; the other five are two SVDQuant operators, GGML k-quants, a mixed `bf16`×FP4 GEMM and a SmoothQuant-folded quantizer. The corpus covers the quantization arithmetic and FP8 KV-cache decode attention, but not the routing or the other quantized attention built on them.

Table 3: The axes a contract can constrain, how each axis is decided, and the corpus's coverage. *Contracts* counts the contracts that declare at least one clause on the axis; a contract declares between one and five axes, 246 clauses in all.

| axis | decided by | clauses | contracts |
|---|---|---|---|
| rounding | membership, or exactness at the tie sites | 29 | 27 |
| saturation | exactness at the clamp sites | 22 | 22 |
| parameter | membership, exactness or a bound on the parameter | 37 | 24 |
| degenerate | exactness at input-determined sites | 12 | 11 |
| storage | exactness in the code domain | 29 | 27 |
| accumulation | a declared bound: reassociation, step or tolerance | 51 | 43 |
| composition | a bound, or membership where a stage returns codes | 60 | 37 |
| structure | a relation, not the values | 6 | 5 |

Table 4: QKERNELBENCH and existing GPU-kernel generation benchmarks. *Integrity*: built-in defenses against reward hacking.

| Benchmark | #Tasks | Language | Quant. tasks | Perf. reference | Integrity | Diagnostic axis |
|---|---|---|---|---|---|---|
| KernelBench (2025) | 250 | CUDA | 0 | PyTorch eager | – | fusion level |
| TritonBench (2025a) | 184 | Triton | $\sim$0 | call-accuracy + speedup | – | channel |
| MultiKernelBench (2025) | 285 | multi-platform | few | platform baseline | – | category/ platform |
| KernelGenBench (2026) | 210+110 | Triton/ multi-chip | few | library/ platform | 3-tier | source/ platform |
| GEAK-eval (2025) | 184+30 | Triton | 0 | reference kernel | – | agent iterations |
| KernelBench-X (2026) | 176 | Triton | 6 (0/30) | eager + efficiency | partial | 15 categories |
| AgentKernelArena (2026) | 196 | HIP/Triton | 0 | tuned baseline | workspaces | unseen configs |
| FastKernels (2026) | 46 archs | CUDA | few | production stack | integration | production fit |
| Robust-kbench (2025b) | subset | CUDA | 0 | eager | verification | soundness |
| FlashInfer-Bench (2026) | 41 (1.6k wl.) | multi-DSL | FP8/FP4 consumers, 0 quantizers | library kernels | closed loop | serving traces |
| SOL-ExecBench (2026) | 235 | CUDA | dtypes only | SOL bound | hardened | SOL gap |
| **QKERNELBENCH (ours)** | 84 | **CUDA** | **80 (+4 HP baselines)** | **PyTorch reference** | **held-out seeds** | **what can go wrong** |

**Sources of the shapes.** Feature dimensions come from released model configurations. In a trace of SGLang serving Qwen3-8B with FP8 weights under $400$ ShareGPT requests, the token count handed to the activation quantizer takes $247$ distinct values, concentrated at the running-request cap ($64$), at $1$ and at the chunked-prefill cap ($8192$), and the powers of two from $128$ to $4096$ that most workload cases use occur $0$ times.

## B CONTRACTS, VERDICT CLASSES AND HOW TO WRITE ONE

**Schema.** A contract's declaration gives the signature, shapes and dtypes; a `numerics` block (rounding, tie direction, saturation endpoints, zero guard, non-finite policy, accumulation dtype, scale formula and storage layout); a trigger for each clause; the domain; and one verdict per output. Cases separate the definition from the workload: a workload case records the model and site it comes from, and a constructed case records the clause it exercises.

**Verdict classes.** A verdict class is chosen by the type of the output.

- *Codes with a returned parameter.* `witness` implements membership. The declaration names the output that carries the parameter, a budget within which the returned scale may differ from the reference's, the admissible paths, and the `exact_at` positions where no freedom is granted; the verdict reconstructs the codes at the returned parameter along each path and accepts if any path reproduces them. The paths are `divide` ($x/s$), `reciprocal` ($x \cdot (1/s)$) and, for quantizers whose scale is $\mathrm{absmax}/q_{\max}$, the third path `qmax_over_absmax` ($x \cdot (q_{\max}/\mathrm{absmax})$), the order that vLLM's and SGLang's INT8 quantizers use; it takes its factor from the input, and the budget still bounds the returned scale. A factor or quotient formed by an approximate division can be off by two `float32` ULPs, so the ten integer quantizers with the third path declare `factor_ulps: 2`: a site whose scaled value lies within that error of a rounding boundary may take either code,

except at an exact tie, where the declared rounding rule decides. The other two contracts with the third path, `fp8_per_token_act_quant` and `mla_kv_latent_fp8_quant`, quantize to FP8 and declare no such band. A path then contributes at most two codes per site where the construction of Section 2 counts one. `mx_witness` does the same for a shared power-of-two exponent.

- *Other codes.* `bit_exact` and `code_exact` compare bit patterns, after unpacking where the output is packed. `fp8_code_step` and `site_conditional` require exactness outside a lawful-site mask computed from the contract and the input (the near-tie band, the saturation boundary, the underflow band), admit one code step inside it, and report the mask's coverage; `fp8_code_step` measures distance in code index, since one step near the top of the range is 32 in value for E4M3 and 8,192 for E5M2. Where no mask is declared, as for the two saturating FP8 casts, the verdict is exact. In the released corpus five code outputs are decided with a mask: those of `apply_row_scale_to_codes` and `fused_rope_fp8_quant`, the touched entries of `kv_int8_paged_quant`, and the zero points of `int8_asym_per_row_quant` and `zero_point_from_minmax`.
- *Floating-point values.* `ulp_bounded` admits $k$ representable steps of the output dtype where the operation order is fixed; `reassoc_bounded` is the order-independent bound of Appendix C; `tolerance_bounded` is an elementwise tolerance, permitted only on dense floating-point value outputs.
- *Structure.* Top-$k$ selection is checked for set membership and then order, and a paged write for touched versus untouched entries.

**The scale budget.** Fourteen contracts declare membership, `mla_kv_latent_fp8_quant` for two outputs, and twelve of them also the third path, all but `int8_asym_per_row_quant` and `nonfinite_policy_quant`. Of the 15 membership verdicts, 13 declare a scale budget, all of two representable steps; the other two, whose quantized value is a computed transform, require the scale to lie within $2^{-18}$ of the declared rule computed exactly (Appendix K.3). Recomputing each contract's scale in `fp64` and rounding back moves it by zero steps on every contract but one, `nonfinite_policy_quant`, where it moves by one, so the budget is at least twice what this lawful alternative needs. The other lawful alternatives are the declared paths, which two release gates check: no undeclared path may change the result, and the allowance must behave as declared. Of the 30 outputs under a step bound, all but one declare $k = 2$.

**Writing a contract.** Every released contract satisfies the 30 release gates of Table 5, which enforce, in order: the reference before the clauses, with each clause checkable from the output; an input condition that makes each clause observable; a verdict chosen by the output's type, with a derived bound where one exists; one case per production shape plus boundary and constructed cases; calibration against mutants (Appendix D); and the reference judged under every lawful alternative that applies.

Table 5: The release gates, each by what it checks; the gates check rules and structure, not every constant or the arithmetic of tensor cores. All pass on the released corpus. The last reruns calibration under other seeds.

| | what the gate checks |
|---|---|
| 1 | A content hash for the corpus, so every result names the corpus it scored. |
| 2 | Can every contract generate its inputs and run its reference? |
| 3 | Every contract declares a verdict, and its own reference passes it. |
| 4 | Every declared clause that depends on the input is reached by some case; a clause that does not says so. |
| 5 | Mutation calibration: can each contract's verdict tell right from wrong? |
| 6 | The verdicts declared in the contracts reproduce the decisions of the earlier specification. |
| 7 | Every declared comparison domain is either honored by the verdict or provably implied by its class. |
| 8 | A case labeled with an edge must not be a copy of the base case. |
| 9 | A constructed case must stay inside the contract's own input domain. |
| 10 | The prose a model reads must be what the reference actually does. |
| 11 | A membership contract may leave division or reciprocal multiplication undeclared only where the two give the same codes on its own cases. |
| 12 | A generator must be a deterministic function of its parameters and seed. |
| 13 | Each verdict accepts the reference computed with a reassociated reduction, a wider accumulator or fused multiply-adds. |
| 14 | The prompt must state the task, not the record of how the task got its rules. |
| 15 | Do the held-out seeds actually change the data? |
| 16 | Every declared clause names an axis in the closed taxonomy, and every axis is covered. |
| 17 | Does any reference implement a rule its contract does not declare? |
| 18 | The MX shared exponent must equal the one integer arithmetic gives, at every power of two. |
| 19 | Every decision procedure is reachable from a contract, or says why not. |
| 20 | A code output must not depend on which device computed it. |
| 21 | No code output is decided by a distance. |
| 22 | No tool may normalize a candidate's outputs more narrowly than the runner. |
| 23 | A judging record must name one corpus, and it must be the corpus in the tree. |
| 24 | The task statement must name the dtype of every output it declares. |
| 25 | Every contract declares the dtype of each output, written from its reference and checked against it. |
| 26 | The rule the scorer runs must not be looser than the verdict the contract declares. |
| 27 | Does each declared scale allowance admit what it should and nothing else? |
| 28 | Does each stage decomposition actually describe its contract? |
| 29 | Release gate: the corpus manifest, the task tree, the contracts and the validator must agree. |
| 30 | Is the calibration a property of the corpus, or of one draw? |

## C An Order-Independent Bound for Reductions

**The bound.** A contract that fixes the accumulation dtype but not the summation order permits every order, so its verdict must admit all of them. For a dot product of $K$ terms whose products are exact in the accumulator (two `bfloat16` values multiply exactly in `fp32`), every order satisfies $|\mathrm{fl}(\sum) - \sum| \le \gamma_{K-1} \sum_k |a_k b_k|$ with $\gamma_n = nu/(1-nu)$ and $u = 2^{-24}$ (Higham, 2002). The candidate $c$ and the reference $r$ are both such sums rounded to the output dtype, with unit roundoff $u_{\text{out}}$, so for any two admissible orders

$$|c - r| \;\le\; 2\gamma_{K-1}\, S + u_{\text{out}}\,(|c| + |r|), \qquad S = \sum_k |a_k b_k| \text{ in } \texttt{fp64}.$$

Strictly, the rounding term carries a factor $1/(1-u_{\text{out}})$, about $1.004$ for `bf16`, which the verdict omits. The derivation also assumes that every addition rounds to nearest. Tensor-core accumulation does not guarantee this, so for a kernel that uses tensor cores the bound is not a proof; one evaluation kernel for the GEMM contract uses `wmma` and satisfies the bound in every cell. The released corpus declares this bound, as the class `reassoc_bounded`, for its two `bf16` reduction contracts, and the variants derived below for `fp8_gemm_pertensor` and `fp8_kv_decode_attention`. The contract names the operands and the accumulation dtype, and the verdict computes $S$ in `fp64` and $\gamma$ from the declared dtypes, with $n = K - 1$ when every product is exact in the accumulator. Its calibration derives split-K over four and eight partitions and `fp64` accumulation as admissible variants, and accumulation in the output dtype and an error of four bounds at $0.1\%$ of positions as defects; both contracts calibrate.

**Effect of the bound.** Under the two-ULP bound it replaces, the `bf16` GEMM contract rejected all 12 kernels that the development population wrote for it, because split-K and `fp64` accumulation move the reference's own result by up to 6 steps. A pass rate of zero does not by itself say whether the kernels or the rule are wrong; here the receipts showed the rule. Under the bound, 11 are accepted, with a largest ratio of error to bound of 0.44, as are the reference's own split-K and `fp64` variants; the remaining kernel violates the bound at 521,493 of 524,288 elements, by a factor of up to 204, and is wrong at 99.5% of its elements. All 12 decidable GEMV kernels satisfy it. Judging the 24 development rows of the two contracts again, with the bound declared as a verdict class rather than applied by the earlier plan, changes 0 decisions.

**Limits of the bound.** Accumulating in `bf16` is rejected at 306,954 of 524,288 elements and in `fp16` at 2,234. Rounding every product to `bf16` before an `fp32` accumulation, or truncating the final rounding, is not rejected: on this case its effect is at most two `bfloat16` steps, inside the spread of lawful reassociation, so no bound on the value can separate it from a lawful order. This limit applies to any value verdict; membership on a code output does not have it.

**Tensor cores and attention.** `fp8_gemm_pertensor` declared two ULPs of its `fp16` output and rejected the FP8 GEMMs of cuBLASLt and SGLang on the Ada card at 3 to 5 per cent of positions. An FP8 tensor core adds each group of exact products with about 13 fraction bits before the `fp32` accumulator, as measured on the Ada card, and with about 14 on Hopper (DeepSeek-AI, 2024), so each group sum is off by up to $2^{-13}$ of its magnitude. The contract declares this, and its verdict is the bound above for the scaled product,

$$|c - r| \le 2\big(\gamma_{K-1} + \gamma_2 + 2^{-13}\big)\,|s_a s_b| \sum_k |a_k b_k| + u_{\text{out}}(|c| + |r|),$$

with $\gamma_2$ for the two scale multiplications. The production GEMMs use at most 54 per cent of it on every case at three seeds, and its calibration adds the group accumulation as an admissible variant. This contract accounted for three of the six cells in which Blackwell and Ada disagreed (Appendix I), and under the bound the kernel concerned passes on every device. `fp8_kv_decode_attention` declared two ULPs of its output, measured from the reference or from its `fp64` evaluation, and rejected every decode path of FlashInfer and SGLang's Triton decode kernel. These kernels round the unnormalized probabilities $e^{s-\max}$ to half precision before the product with $V$, as FlashAttention does, and the output, a weighted sum of value rows of both signs, cancels. With $p$ the exact probabilities, $n_k$ the number of keys, $D$ the head dimension, $L$ the row's largest $\sum_d |q_d k_{sd}|/\sqrt{D}$, which bounds how far an `fp32` logit can move, and $u_h = 2^{-11}$, the verdict is

$$|c - r| \le 2\big(\gamma_{n_k} + 2\gamma_D L + u_h\big) \sum_s p_s |v_s| + u_{\text{out}}(|c| + |r|),$$

plus what probabilities below the half-precision normal range can contribute. The verdict's receipts show that FlashInfer's kernels use at most 28 per cent of it and SGLang's 76. Its calibration adds the rounded probabilities and eight-way flash-decoding as admissible variants and an `fp16` accumulator for the product with $V$ as a defect, and the reference's split-K variants reach it through the gate above.

## D Calibration and an End-to-End Check

**Calibration.** The calibration tool generates, from each contract's declaration, defects that break its clauses and lawful variants for its declared freedoms, 225 and 93 over the corpus, and requires every defect to be rejected and every lawful variant to be accepted. A clause with no mutant that the verdict can see is reported as INERT, and a verdict that rejects a lawful variant as OVER-STRICT; either outcome fails the release. The mutants come from the declaration, so they test what a contract declares and nothing it leaves out; the gates on triggers and the audit of prompt constants (Appendix J) cover part of that gap. All 84 contracts calibrate against 318 mutants, and a gate re-runs the whole calibration at four seeds.

Membership separates the 8 variants it applies to. The lawful-site mask admits one defect, a quantizer that breaks ties away from zero, on a contract whose mask declares near-tie positions free to move by one step. The defect acts only at tie positions, the mask covers 0.026% of positions, and

every position the defect affects falls inside it, whereas membership rejects it immediately. A contract that declares a freedom wider than the operator has will admit a defect that falls inside it. In the released corpus five code outputs are decided with a mask (Appendix B); the calibration of `apply_row_scale_to_codes` includes this mutant and rejects it on four of its five cases. FlashInfer-Bench's $5\%$ allowance is better than a uniform bound, and every defect it admits is a systematic error confined to a small subpopulation (the saturation clamp, both tie-direction defects and the missing zero guard). Such errors are common in quantizers, since a saturation rule governs only saturated elements and a tie rule governs only ties. Cosine similarity fails for another reason: it is scale-invariant, so it cannot see a wrong scale, and it admits truncation instead of rounding, which is wrong at every element but barely rotates the vector.

**End-to-end cost of a tensor-level defect.** We substituted quantizer variants into all $154$ linear layers of TinyLlama-1.1B and measured perplexity on one wikitext-2 slice (Table 6).

Table 6: Each variant substituted into every quantized linear layer of one model, with the resulting perplexity.

| variant | uniform $\mathrm{atol}=1$ | integrity | PPL | $\Delta$ vs. reference |
|---|---|---|---|---|
| reference (INT4, per row) | pass | clean | 11.398 | — |
| correct: reciprocal expression | pass | clean | 11.389 | $-0.009$ |
| defect: clamp edge (observed) | pass | clean | 11.398 | $+0.000$ |
| defect: tie direction (observed) | pass | clean | 11.401 | $+0.003$ |
| defect: truncate, not round | pass | clean | **6648.1** | $\mathbf{+6636.7}$ |
| cheat: return input unquantized | pass | **flagged: lattice** | 8.514 | $-2.884$ |

A uniform tolerance of one code admits truncation, which destroys the model. Perplexity is no substitute either: the two defects observed in generated kernels cost $+0.000$ and $+0.003$, and the cheat that skips quantization has the best perplexity in the table because it is the unquantized model. The three checks fail on different classes of defect, and only the unit contracts, whose inputs are constructed at the contested positions, catch the clamp and tie defects. A scheme with little headroom cannot expose such a defect end to end, so the absence of end-to-end damage is not evidence of correctness.

## E The Evaluation Campaign in Detail

**Client and prompt.** Every model is reached through one API gateway and named by the identifier the gateway serves. Each kernel comes from one request with no tools. The harness renders each prompt and extracts each kernel, and every prompt sent is checked to be byte-identical to the one the harness renders. The prompt contains the contract but not the reference: the specification, the argument names of the signature, the source of the input generator, every numerics clause, the domain, a per-output acceptance criterion, and a build-environment section (the compiler flags in force, the header each CUDA and `torch` symbol requires, that no helper macros are provided, and the return types the binding accepts). It asks for one CUDA source file in a single fenced block and states that no feedback will follow. The prompts were rendered from the corpus as it stood before any change of Table 12.

**Budget and repeated calls.** No extended-thinking parameter and no sampling parameter is sent, so every model is called alike and samples at its provider's default; every model still spends reasoning tokens under the gateway's default. The output budget was 32,000 tokens, and 45 calls, 37 from GLM-5.3 and $8$ from DeepSeek-V4-Pro, spent all of it on reasoning without emitting code. Each of these was sent once more with 120,000 tokens, and the paper reports that answer, so these two models received a second draw that the others did not, and a pass rate under a fixed budget partly measures the budget. A request that yields no kernel counts as not built.

**Judging.** Each kernel is compiled once, in its own process, and run on every declared case at the visible seed and at seeds $1337$ and $987654321$, with inputs generated on the GPU as the harness generates them; since these seeds are now published, a later evaluation should draw fresh ones,

which the generators allow. The contract's own case at the visible seed runs first, so that a fault on a later case cannot take the own-case verdict with it. A kernel whose process dies is re-run on that case alone; both kernels for which this happened died again and have no verdict on that case. In each cell the contract and the five rule settings judge the same outputs after the shared interface layer (Appendix F).

**Build failures.** We rebuilt every attempt without a verdict, 139 in all, and classify each by the first message of the compiler or linker: 15 build on the rebuild and so failed at run time, and 3 use `CUDART_INF_F`, whose header the prompt's table omits, the only failures caused by an omission in the prompt. The rest are the models' errors, most often a call to `data_ptr<__half>()`, which PyTorch does not instantiate (32 kernels), or a CUDA symbol that does not exist (26); the two Qwen3.5 models account for most of them.

**Output dtype.** Of the six interface errors, four return FP32 where the prompt states FP16, on the FP8-named INT8 contract of Appendix J; three of them are right in value (Appendix H), and we count all four as errors because the prompt states the dtype.

**Difficulty of the contracts.** Of the 84 contracts, 50 are solved by all six frontier models, and every contract by at least three; across both rounds, 33 are solved by all 6 frontier models of Table 15 each time (Appendix N). Under the campaign version, 49 were solved by all six and three by none, all of them repaired afterwards. The corpus therefore tests conformance to declared numerics better than it ranks frontier models, which Table 1 reports from one draw each.

**Limits of own-case acceptance.** Of the 6 kernels that the contract accepts on the own case but that are not solved, 5 fail a constructed boundary case and 1 fails only another workload shape or a held-out seed; each failure's receipt names its verdict and positions, and the case names the clause it was built for. Under the campaign version there were 28 such kernels. Reading their receipts led to the defect analysis of Appendix K.3, which found 22 of them correct and rejected by defects of their contracts.

**Speed.** The 501 solved kernels were timed on an RTX 6000 Ada, the Ada card, one at a time on an otherwise idle card, each in a fresh process on the contract's own case at the visible seed, with CUDA events around 50 iterations after 10 warm-up runs, against the PyTorch reference that the contract ships; every timing in this appendix was taken while no other process used the card. The same seed draws different inputs on different GPUs (Appendix I), so every timed kernel was also judged on the inputs it was timed with. Of the 501, 500 are accepted there and form the speed column of Table 1. The exception, Claude Sonnet 5's `int8_asym_per_row_quant` kernel, is rejected on the Ada card in every cell and accepted on the RTX PRO 6000 in every cell; it reloads each row's scale and zero point after a barrier through `__restrict__` pointers, and without those qualifiers it is accepted on Ada too. In the $\mathrm{fast}_p$ measure of KernelBench, the share of the 84 contracts that a model solves with a kernel more than $p$ times as fast as the reference, the six frontier models reach $\mathrm{fast}_1$ between 73% and 86% and $\mathrm{fast}_2$ between 58% and 73%. The per-model means in Table 1 cover different sets of contracts, so they are not directly comparable.

**Against `torch.compile`.** The reference is plain PyTorch, so we also compiled every contract's reference with `torch.compile` (default mode, static shapes, torch 2.9) and timed it on the same card and case. All 84 compile. The compiled reference is a geometric-mean $1.18\times$ faster than the eager one and slower on 43 contracts, most of them references that run in under 0.2 ms. Its output fails the contract on 10 contracts; all but one are quantizers or FP8 casts, and on them at most $0.5\%$ of its codes differ from what the declared arithmetic produces.

**Against shipping kernels on the own case.** In the Ada environment where the evaluation's kernels were timed, our own-case timing harness calls the shipping kernel of nine contracts, each computing the contract's operation on its operands, and six of them have one that runs and that the contract accepts on its own case: cuBLAS for the INT8, `bf16` GEMM and GEMV contracts, cuBLASLt and SGLang for the FP8 GEMM, and SGLang for the INT8 `scaled_mm` and the FP8 per-token quantizer (Appendix G). We timed that kernel on the contract's own case. On these 6 contracts, each frontier model's kernels accepted on their timing inputs reach a geometric mean of

11% to 27% of the shipping kernel's speed, over the contracts among them that it solves, and 6 of the 36 generated kernels on them match or exceed it, all on the bandwidth-bound GEMV. On the INT8 and FP8 GEMMs the median generated kernel reaches under 5%.

## F THE PUBLISHED RULES, AS IMPLEMENTED AND COMPARED

**The rules.** Each rule follows its source where one exists. *KernelBench* applies `torch.allclose` to every output with `rtol` $=$ `atol` $= 10^{-2}$ after casting to float, which is what a harness written for dense kernels does with integer and FP8 outputs (Ouyang et al., 2025). *KernelBench-X* requires a cosine similarity of at least 0.95, a relative $L_1$ of at most 0.05 and an RMSE over the mean magnitude of at most 0.10, all jointly, the strict end of its published ranges (Wang et al., 2026). In *FlashInfer-Bench*, an element exceeds the tolerance when it fails both $|x - y| \leq$ `atol` and $|x - y|/(|y| + 10^{-8}) \leq$ `rtol`, with `rtol` $=$ `atol` $= 10^{-2}$, and an output passes when the fraction of elements within tolerance reaches the required matched ratio; the shipped configuration leaves the ratio unset, which resolves to 1.0, while its paper reports 0.95 for low-bit outputs (Xing et al., 2026). The *calibrated tolerance* follows Sarkar (2026), whose corpus has no code outputs: each output's tolerance is the largest deviation of the reference from an `fp64` recomputation of itself over three draws. We add a floor, one `float32` step at the output's largest magnitude and one for an integer output, and compare every element against this single tolerance. KernelBench's rule and the calibrated tolerance were not written for codes; how we apply them to codes is our choice, and the paragraphs below measure what those choices cost.

**The shared interface layer.** As shipped, each rule handles rank and dtype itself, and that handling accounts for part of what a naive comparison would attribute to the rule. Every comparison in this paper therefore runs the contract's interface layer first. It reconciles rank only: a per-row scale returned with shape $(T, 1)$ where the reference returns $(T)$ is reshaped for every rule, and a kernel that returns the wrong number of outputs, or an output whose bit width differs from the one the task names or whose size differs from the reference's, is rejected by every rule. Outputs are otherwise compared as returned: INT4 codes as the packed words the kernel returns, and an FP8 code returned as its raw byte as a number, so such kernels are counted apart in Table 2. On the 586 kernels of the development campaign, reading the rules as shipped would have added 11, 7 and 4 wrongful acceptances and 14, 6 and 10 wrongful rejections to KernelBench-X, KernelBench and the calibrated tolerance respectively, all of them caused by interface handling.

**The construction, measured.** Table 7 applies the distances of Section 2 to its two candidates.

Table 7: Two candidates on $x = [0.5, 1.5, 127]$ against $q_{\text{ref}} = [0, 2, 127]$. The first is correct under a scale one ULP from the reference's; the second is produced by no admissible scale. Every distance that reads only the codes is identical to the printed precision, and cosine similarity differs only in the eighth decimal place, whereas published cosine thresholds lie between 0.90 and 0.95. Membership separates the two candidates.

| | correct, $\theta = \theta'$ | wrong | separable? |
|---|---|---|---|
| elementwise $\lvert q - q_{\text{ref}}\rvert$ | $[0, 1, 0]$ | $[0, 1, 0]$ | no |
| $L_1$, $L_\infty$ | 1.000000 | 1.000000 | no |
| RMSE | 0.577350 | 0.577350 | no |
| relative $L_1$ | 0.0077519 | 0.0077519 | no |
| cosine | 0.999969009065 | 0.999969024429 | $1.5 \times 10^{-8}$ |
| KernelBench `allclose` | **rejects** | rejects | — |
| KernelBench-X | accepts | **accepts** | — |
| membership | accepts | rejects | **yes** |

**On every case.** Table 2 evaluates each rule on the contract's own case. Over every declared case at three seeds, counting a kernel as accepted by the contract, or by a rule, only if it is accepted on every cell with a verdict (only the two kernels whose process died, Appendix E, have cells without one), every setting still disagrees with the contract (Table 8): the strict settings reject about a hundred

correct kernels each, and the two loose settings reject about as many correct kernels as they accept wrong ones.

Table 8: The comparison of Table 2 on the evaluation campaign, on the contract's own case and over every case at three seeds. Under the campaign version the 22 correct kernels that its defects rejected, and the one that the shipping-kernel repair later admitted, counted as wrongful acceptances for each rule that accepted them on every cell, so that KernelBench-X accepted 27 wrongly, and not against the rules that rejected them, so that `allclose` rejected 87 wrongly (Appendix K.3).

| | own case | | every case, three seeds | |
|---|---|---|---|---|
| rule | acc. wr. | rej. wr. | acc. wr. | rej. wr. |
| FlashInfer-Bench, ratio 0.95 | 6 | 6 | 9 | 9 |
| KernelBench-X | 6 | 0 | 8 | 12 |
| calibrated tolerance | 3 | 70 | 5 | 100 |
| KernelBench `allclose` | 2 | 58 | 2 | 108 |
| FlashInfer-Bench, ratio 1.0 | 2 | 59 | 2 | 108 |

**What each rule does to each model.** Table 9 applies each rule setting to every kernel of each model on every declared case at three seeds, in both rounds, and counts the contracts it would report as solved. Under `allclose` and FlashInfer-Bench at its default ratio each frontier model loses more contracts than the frontier models differ from one another under the contract; the loose settings stay close to the contract's count because their wrongful acceptances and rejections nearly cancel.

Table 9: What each rule setting does to each model's result: contracts solved on every declared case at three seeds under the contract and under each rule setting, in the first round / the second (Appendix N). A rule solves a contract when it accepts the kernel on every cell with a verdict, while the contract requires every declared case, so the two kernels whose process died on one case can count for a rule and not for the contract; kernels that return raw storage bytes are counted as the contract counts them, as in Table 2.

| model | contract | `allclose` | KernelBench-X | FIB 0.95 | FIB 1.0 | calibrated |
|---|---|---|---|---|---|---|
| Claude Sonnet 5 | 72/71 | 54/55 | 72/72 | 75/74 | 54/55 | 59/60 |
| GPT-5.6-sol | 77/76 | 60/59 | 76/74 | 75/75 | 60/59 | 62/61 |
| Kimi-K3 | 80/76 | 62/60 | 79/75 | 79/76 | 62/60 | 62/59 |
| DeepSeek-V4-Pro | 71/69 | 55/55 | 69/68 | 69/68 | 55/55 | 58/55 |
| GLM-5.3 | 79/72 | 63/57 | 77/70 | 77/72 | 63/58 | 63/57 |
| Qwen3.8-Max | 79/80 | 63/64 | 77/78 | 79/80 | 63/64 | 65/67 |
| Qwen3.5-397B-A17B | 39/41 | 34/35 | 42/43 | 42/42 | 34/35 | 33/36 |
| Qwen3.5-35B-A3B | 4/4 | 5/5 | 5/5 | 5/5 | 5/5 | 5/4 |

**Distribution of the errors.** Wrongful rejections concentrate on contracts whose verdict grants a declared freedom: a scale path on the INT4, INT8 and FP8 quantizers, and a summation order on the reductions; for the calibrated tolerance, most of them are the floor effect described below. KernelBench-X's 6 wrongful acceptances are Claude Sonnet 5's FP4 decoder; two Qwen3.5-397B-A17B quantizers, INT8 per group and INT4 per channel, whose codes no declared path reproduces at a few positions; a Qwen3.5-35B-A3B scale kernel that applies its zero guard before the division; and two `w8a8_linear` kernels, one of them holding the row scale in fp16, whose outputs miss the declared tolerance while their aggregate distance stays within KernelBench-X's thresholds. FlashInfer-Bench at 0.95 accepts the first four as well, and instead of the two `w8a8_linear` kernels two FP4 encoders that produce the negative-zero code their contract excludes; its 6 wrongful rejections are five `llm_int8_linear` kernels and one `bf16` GEMM, where more than five per cent of the outputs of a correct kernel lie outside $\texttt{rtol} = \texttt{atol} = 10^{-2}$.

**Reading the parameter without exactness.** Membership differs from a distance in two respects at once: it reads the returned parameter, and it compares codes exactly. We separated the two on the

development campaign by recomputing the reference at the candidate's returned parameter along each declared path and then applying KernelBench's `allclose`. On the contract's own case this removes every one of KernelBench's 19 wrongful rejections on the 62 single-membership kernels without admitting anything. Over every case and seed it removes 24 of 30 and accepts 10 kernels that membership rejects, because a tolerance around a reconstructed reference readmits the one-code discrepancies that a declared rounding or tie rule forbids. On the evaluation campaign we applied the same rule to the 81 kernels of the 13 contracts that judge exactly one output by membership, comparing codes in the domain the witness reads them (INT4 unpacked). On the contract's own case it agrees with every verdict of the contract, where KernelBench's rule rejects 47 of these kernels wrongly. Over every case and seed it rejects 10 kernels that the contract accepts, against 61 for KernelBench's rule, and accepts none that the contract rejects. The 10 are kernels of `layernorm_quant` and `hadamard_quant_int8`, whose repaired witnesses reconstruct from the `float64` value of the declared transform; a kernel that computes the transform in `float32` can differ from that reconstruction by one code where the value lies within the declared budget of a rounding boundary, which the witness admits and an exact comparison of codes does not. Reading the returned parameter thus accounts for most of the contract's advantage over KernelBench's rule on these contracts, and admitting the lawful moves of a computed tensor for the rest. Under the campaign version the rule rejected three kernels, all on `mla_kv_latent_fp8_quant`, which judges two outputs by membership and is not included.

**The calibrated tolerance without our choices.** Where the `fp64` recomputation, which the reference returns in its own output dtype, reproduces a low-precision output exactly, as it does for the `bf16` GEMM and the FP8 quantizers, the tolerance is our floor, far below one step of a `bf16`, half or FP8 output. On the evaluation's own cases, 46 of the rule's 70 wrongful rejections are of this kind (40 GEMM, GEMV, convolution and attention outputs and 6 FP8 codes). With the floor raised to one step of the output dtype at the output's largest magnitude, it rejects 24 correct kernels, all INT4 quantizers, and still accepts 3 wrong ones; it then also accepts SGLang's FP8 quantizer on the four cases where its codes differ from the reference's (Appendix G). The remaining rejections come from the packed INT4 words, where a one-code difference in the high nibble of a word is a difference of $2^{28}$ in the word's integer value, far above a floor of one. To remove this second choice too, we re-ran the 26 decidable INT4 kernels of the evaluation on the Ada card, with inputs drawn there, and judged each by the contract and by the calibrated tolerance twice. As Table 2 applies it, on packed words with a `float32` floor, the rule rejects 24 of the 25 kernels the contract accepts, as on the judging card. Reading the codes unpacked through the contract's own domain, with floors of one code step and one step of the scale's dtype, it rejects none of them, since no accepted kernel differs from the reference by more than one code, and it accepts the one kernel the contract rejects. Combining these Ada verdicts with the dtype floor on the other contracts, the rule accepts 4 wrong kernels on the evaluation's own cases and rejects no correct one: without our two choices it is a loose setting like KernelBench-X. On the development campaign, widening the calibrated tolerance, with its `float32` floor, to the largest deviation that any lawful reassociation of the reference produces halves its wrongful rejections on the contract's own case (61 to 30) and recovers only 4 of 67 over every case, because on a code output the tolerance is the floor of one, so its width is not the issue.

On the evaluation's own cases, the 95% interval for the difference between a rule's wrongful rejections and its wrongful acceptances is $[-12, -1]$ for KernelBench-X, and $[26, 90]$ and $[34, 103]$ for `allclose` and the calibrated tolerance, excluding zero in each case, toward acceptance for KernelBench-X and toward rejection for the other two. KernelBench-X accepts every kernel that `allclose` or the calibrated tolerance accepts, and more, while those two are incomparable, each accepting kernels that the other rejects; the development population shows the same containments. At its default ratio, FlashInfer-Bench decides as `allclose` does on 529 of the 530 kernels, and at 0.95 it is one of the two loosest settings, so its outcome depends on the ratio more than on the metric.

## G Shipping Kernels and Production Paths

**Procedure.** We ran every shipping kernel that we could run and that computes the operation of one of our contracts on the same operands, unmodified, on every declared case at the visible seed and the two held-out seeds: SGLang, FlashInfer, cuBLAS and PyTorch on the Ada card, the GB10

and the RTX PRO 6000 (28 of 34 implementations on 17 contracts accepted on every case they run), then vLLM, torchao, DeepGEMM and TensorRT-LLM (40 of 62 on 36 contracts). Layouts and scale conventions are converted outside the call; a scale returned in another floating-point dtype is widened and judged by value, since a shipping kernel follows its library's interface rather than a prompt. Kernels that need GPUs we do not have (vLLM's Machete, DeepGEMM's Hopper and data-center Blackwell kernels) are not judged, and DeepGEMM's `sm_120` path, which takes only power-of-two scales, is judged on the contract's inputs with the scales rounded up to powers of two and the reference computed on the same inputs.

**What the first pass changed.** Under the campaign version the first pass rejected correct outputs of three kinds, and the repaired release changes the verdicts responsible (Appendix K.4): the tensor-core FP8 GEMMs of cuBLASLt and SGLang, which exceeded a two-ULP bound (Appendix C); FlashInfer's and SGLang's decode kernels, whose half-precision probabilities exceeded another; and the INT8 quantizers of vLLM and SGLang, which form their factor as $q_{\max}/\mathrm{absmax}$, a third path the contracts did not declare. A second-pass contract, `zero_point_from_minmax`, admitted only one of two readings of a one-sided range; the released corpus accepts torchao's reading as well, under which it is accepted on all $24$ cells.

**What the contracts still reject.** Under the released corpus every other rejection has one of three causes. Most follow another convention for a stated clause: truncation or ties away from zero where the contract rounds half to even, a zero scale or a floored absmax on all-zero rows, a row's own range where the contract includes zero, saturation at $-128$, division by $128$, a sign-extended INT4 unpack, an unordered top-$k$, a `float16` scale where the contract declares `float32`, or $+0.0$ for FP4 code 8. Some compute in a lower precision than the contract admits: torchao's affine quantizer divides in the input's `float16` on seven contracts, and the GPTQ GEMM that SGLang and vLLM ship accumulates in half precision; both precisions were stated in the contracts only after these rejections, with no verdict changed (Appendix K.5). And the MX quantizers of FlashInfer 0.6.1 and TensorRT-LLM return codes on the Ada card that change between two calls on the same input. Companion contracts outside the released corpus that state SGLang's conventions accept its truncating group quantizers on all $24$ cells on each device.

**The rules on a shipping quantizer.** On 4 of the 6 cases on which SGLang's FP8 per-token quantizer is accepted, its codes differ from the reference's at $65$, $84$, $768$ and $65$ positions, out of between $917{,}504$ and $16{,}777{,}216$, each time by one E4M3 step of $32.0$. Its scale differs from the reference's in 55 of the 128 rows of the first case, within the declared budget, and membership reproduces every code at the returned scale through the reciprocal path. On these cases KernelBench's `allclose` and FlashInfer-Bench at its default ratio reject the kernel, as does the calibrated tolerance with its `float32` floor; FlashInfer-Bench at $0.95$ accepts it, and so does the calibrated tolerance with a floor of one E4M3 step.

## H Checking the Verdict Layer from Outside

**A second implementation.** A checker written separately from the adjudicator, sharing none of its code, applies each contract's declared rule to the same outputs: byte equality after unpacking with its own unpacker, ULP distance computed from `nextafter` rather than from a relative epsilon, FP8 distance in code index, and the order-independent bound derived from the rule alone. A verdict class it does not implement is declined rather than approximated. For membership outputs, an exact model rederives the reconstruction using `float64` quotients rounded to binary32 under IEEE round-to-nearest-even, the E4M3 encoding taken from the format's tables, and exact rational arithmetic wherever `float64` cannot certify the binary32 rounding; an exact block-format model does the same for MX outputs. On the 586 decidable kernels of the eight-model development campaign, every output has an independent reading: $571$ agree with the contract, $1$ is a rejection that it confirms through a failing output it reads, and $14$ disagree, each on the rank of a returned scale that the checker, which enforces shape and dtype strictly, does not reconcile, and none on arithmetic. Both exact models read every row of each case and every MX block; both reproduce the contract, and on the one MX row that the contract rejects, the model finds the same $38$ violating sites out of $524{,}288$.

**On the evaluation campaign.** The checker and the exact models implement the campaign version's declarations, so they check our implementation of the declarations, not the repairs of Appendix K; on the 530 decidable kernels they are compared with the campaign version's own-case verdicts, which the repairs change for one kernel. They read every output of 504: 493 agree with the contract, 8 disagree on the rank or layout of a returned tensor, which the checker does not reconcile, and 3 on the output dtype of the FP8-named INT8 contract (Appendix J), which the contract's interface layer enforces and the checker, reading a tolerance output by value, does not; the fourth kernel that returns FP32 there is also wrong in value, and the checker rejects it too. None disagrees on arithmetic. The other 26 return packed INT4 codes judged by membership, which neither exact model reads.

# I Whether a Verdict Depends on the Machine

Two devices may lawfully disagree about a floating-point reduction by a ULP. If they disagree about a code that a contract requires exactly, the correct answer depends on where it was computed.

**The development population on three devices.** Every decidable kernel of the development population, 716 over 82 contracts, was judged on every declared case at three seeds on the Ada card, the RTX PRO 6000 and the GB10; the passes on the Ada card and the RTX PRO 6000 have 9,330 cells in common. The kernel, the reference and the verdict run on the GPU, while the inputs are drawn on the host and then moved, because a seeded generator draws a different stream on each GPU; every cell records a hash of its inputs, so that comparability is checked. The Ada pass was run under two framework versions. The pass predates the corpus revision of Appendix K.2, so 13 of its kernels, those of the four INT4 quantizers and `llm_int8_linear`, were judged under their declarations before the revision; none of the cells in which verdicts differ belongs to those contracts. It also predates the repair of the seven defective contracts, so their kernels were judged under their declarations before it, which the revision had not touched.

Table 10: Cell-level comparison of four judging passes. A cell is compared only when its input hashes match, and a cell that ran on only one device is counted separately and never as a disagreement. The two Ada passes have 9,365 cells in common, and their percentage is of those.

| pair | inputs identical | decided on both | verdict differs |
|---|---|---|---|
| Ada vs. Blackwell, same framework build | 100% | 9,316 | 6 |
| Ada vs. GB10, same framework build | 44.7% | 4,154 | 2 |
| Blackwell vs. GB10, same framework build | 44.7% | 4,154 | 0 |
| Ada vs. Ada, two framework versions | 53.6% | 5,010 | 9 |

**Cross-device differences.** Across architectures on a matched build, the verdicts differ in 6 cells on three contracts: `fp8_gemm_pertensor` (3), whose two-ULP bound on a reduction with free order the order-independent bound replaced (Appendix C); `hadamard_quant_int8` (2), where one device admits a scale through the `divide` path; and `fp8_kv_decode_attention` (1), one site of 32,768 under the same kind of bound. None is a disagreement about a code that the contract requires exactly. A framework upgrade on one machine changed more verdicts, 9, all on `layernorm_quant`, whose budget sat on a rounding boundary and whose verdict moved between a single violating site and acceptance, in both directions depending on the case and seed. `hadamard_quant_int8`, `fp8_kv_decode_attention` and `layernorm_quant` are among the contracts repaired later (Appendix K.3); a verdict that changed with the device or the framework build on the same kernel was one sign of their defects.

**The evaluation campaign judged again, and on a GB10.** We built and judged every kernel of the evaluation campaign again on the RTX PRO 6000, with an independent build, and on a GB10, each time on every declared case at three seeds with inputs drawn on the GPU, and then timed the solved kernels on the GB10; the generated kernels' timings on the RTX PRO 6000 are not used. On the RTX PRO 6000, all 530 decidable kernels receive the same own-case verdict as in the evaluation, the solved set is the same 501 kernels, and of the 7,548 cells with a verdict in both passes, 7,543 agree; the five cells that differ belong to one Qwen3.5-397B-A17B kernel whose result changes from run to

run and which is solved in neither pass. On the GB10, whose inputs differ, all 530 own-case verdicts again agree, and one kernel fewer is solved (500), because Claude Sonnet 5's `w8a8_linear` kernel fails the exact zero-row clause on one cell, drawn on the GB10 from the same seed as a Blackwell cell that it passes. Under the campaign version the GB10 solved one kernel more than the RTX PRO 6000, Kimi-K3's `fp8_kv_decode_attention` kernel, which failed one Blackwell cell at two of 32,768 positions under a two-ULP bound and which the repaired contract solves on both.

## J How Much Each Verdict Admits, and What Each Prompt States

Calibration certifies that a verdict rejects the mutants derived from its declaration. It cannot certify that the verdict is tight, because a bound can be loose enough to admit a subtly wrong kernel while still rejecting every mutant. We therefore measure every declared bound in two units that do not depend on its author: the number of representable steps of the output dtype that it spans, and the bound as a fraction of the output's own median magnitude on the contract's case. The measurement reads only the declaration and the reference.

**Outputs without a tolerance.** Of the 109 declared outputs, 83 are decided without a constant in the output's own units: by bit or code equality, by membership, by $k$ representable steps, by a structural or shared-exponent check, or by the order-independent bound. The other 26 carry a tolerance, for two scales a relative one, all of them on floating-point value outputs; no code output anywhere in the corpus carries a value tolerance, and a gate enforces this.

**Loose tolerances.** Ten outputs have a bound above five per cent of their median magnitude, all in matrix-multiply contracts whose `allclose` constants were set by hand or from measurements (Table 11). The worst, `fp8_e4m3_gemm_tile128`, admits 3.4 times the median. On the campaign version we also asked, for every tolerance, whether a kernel returning all zeros would satisfy it at every position, and none does; before the corpus revision the tolerance of `llm_int8_linear` did, because its reference was too small (Appendix K.2). `w8a8_linear`'s tolerance applies around the interval of outputs that the free reduction order and scale path can produce, so its entry, measured around the reference alone, understates what it admits.

Table 11: The declared tolerances that admit more than five per cent of their output's median magnitude. *Level*: the corpus tier (2 low-precision compute, 3 multi-op, 4 fused pipeline). *Kernels*: judged development kernels; *accepted*: those the contract accepted; *margin used*: the largest deviation that an accepted kernel produced, as a fraction of its bound. Every entry is recomputed from the recorded measurements.

| contract | level | bound / median $\lvert y\rvert$ | kernels | accepted | margin used |
|---|---|---|---|---|---|
| `fp8_e4m3_gemm_tile128` | 2 | 3.37× | 3 | 3 | 0.004 |
| `w8a8_gemm` | 2 | 1.33× | 6 | 5 | 0.000 |
| `w4a16_marlin_prepacked` | 3 | 0.219× | 3 | 2 | 0.009 |
| `quantized_residual_block` | 4 | 0.198× | 5 | 5 | 0.005 |
| `llm_int8_linear` | 4 | 0.151× | 3 | 3 | **0.433** |
| `nvfp4_gemm_twolevel` | 2 | 0.083× | 11 | 11 | 0.010 |
| `w4a16_fused_marlin` | 3 | 0.057× | 5 | 5 | 0.032 |
| `gptq_linear` | 4 | 0.055× | 5 | 5 | 0.149 |
| `awq_linear` | 4 | 0.053× | 6 | 6 | 0.325 |
| `w8a8_linear` | 4 | 0.052× | 4 | 2 | 0.000 |

**Slack in practice.** Over the development population, 58 kernels were accepted under a value tolerance, one of these ten or a tighter one. The largest deviation any of them produced is 43% of its bound, and only five exceed a tenth of it, so every one of these tolerances could be tightened twofold without changing a development verdict. On the evaluation campaign, all 56 kernels that these contracts accept on their own case, measured the same way on the Ada card, use at most 44% of their bound, except one `w8a8_linear` kernel at 97%, measured from the reference, which overstates what an interval tolerance uses; nine exceed a tenth of it. We leave the bounds in place, since tightening them could reject kernels that the campaign accepted; extending the order-independent bound to a quantized product is planned for the next corpus version.

**Constants in the prompt.** A rejection is fair only if the convention it names was in the task the model received. Every numeric constant in a contract's `numerics` block (a guard floor, an epsilon, a saturation limit) is something the kernel must implement, so the prompt must state it; a verdict's own tolerance is excluded, since a correct kernel passes whether or not it was told the number. Of the 84 prompts, 80 state every such constant and 4 omit one, in each case the same $10^{-10}$ zero guard. The audit reads the rendered prompt; the descriptions alone omit 18 constants, and the renderer's declared-numerics section supplies 14 of them. The audit compares the prompt with the contract's numerics block, not the block with the reference; they disagreed in one contract, `int8_per_row_quant`, until its repair (Appendix K.3). A scan of all 84 contracts of the campaign version for floors and epsilons that the declaration and the reference state differently finds three differences; two are constants that a declaration mentions in a note, and the third is this one. We also examined every rejection in the development population, as first judged, on a contract whose description omits a constant, 33 in all, by reading what the candidate returned where the reference sits at the omitted constant. Of these, 32 produce no element at that value and were rejected on an unrelated clause, and the remaining one returns a scale of exactly zero, which no guard value produces and which leaves the downstream division undefined, so its rejection does not depend on which guard the prompt states. In the evaluation campaign, the four contracts whose prompt omits the guard have one rejection on the contract's own case, an output of the wrong arity, and two kernels that fail another case; every cell they fail is a random workload or a tie-site case, never a case with an all-zero row, so no verdict there depends on the guard.

**Tie direction.** Where codes are compared exactly, the tie convention decides every position that lands on a half: `torch.round` rounds halves to even, whereas CUDA's `roundf` rounds them away from zero. Of the 21 contracts that declare a tie rule, 20 require one convention and one declares both lawful, and every one of the 21 rendered prompts states what its verdict requires. The audit checks that the rule is stated, not that nothing contradicts it: the prompt of `int4_per_row_quant` also describes a one-code step at a tie as a convention, which its verdict does not allow (Appendix M).

**Names and data.** Of the 49 contracts whose name contains a dtype, two have no tensor of that dtype on either side of the operator. `fp8_gemm_blockscale` and `fp8_grouped_gemm_moe` are named for FP8, and their prose repeats this, while their generator, which is included in the prompt and is authoritative, produces INT8 tensors. We count this as a defect of the task statement and did not repair it. No kernel of the evaluation returns FP8 on either contract; the four kernels on `fp8_grouped_gemm_moe` that return FP32 where the prompt states FP16 are counted as the kernels' interface errors (Appendix E), and every rejection on that contract is of this kind.

# K Every Change to the Corpus After Judging

We changed contracts after the evaluation's kernels were judged, and Appendix N reports a second round judged by the frozen contracts. This appendix gives every version of the corpus, then each change in the order of Table 12: the revision of five contracts, the repair of seven defects that reading the rejected kernels revealed, the repair that the shipping kernels prompted, and the precisions stated without a change of verdict. It ends with what the changes did to the results. No change turned an accepted kernel into a rejected one, and the records of the earlier versions will be released beside the repaired ones.

## K.1 The versions of the corpus

Four versions appear in this paper (Table 13). The content digest covers every contract's files, the path digest the same files with their paths. The models' prompts were rendered from the first version, before the campaign was judged. The revision of five contracts made the second, the *campaign version*; every other contract is byte-identical between the two, so their prompts are the ones the campaign version renders. The evaluation's records were judged under the third, the *repaired release*. The fourth, the *released corpus*, differs from the repaired release in eight contracts: seven gained a statement of the precision of their scale's division, which the adjudicator does not read, and `zero_point_from_minmax` gained an alternative reference, which the adjudicator consults only when the judgment against the reference fails; no record of a generated kernel holds such a failure. A check

Table 12: Every change to a contract after we judged the evaluation's kernels, in order. *Contracts*: those the change touched; *new*: those no earlier row touched, 21 in all. Of them, 16 rejected a correct output we recorded (a generated kernel, a shipping kernel or a constructed variant): every new contract of the first two rows and four of the third, whose other three are quantizers that gained the third path with their siblings. *Recorded kernels*: kernels the change turned from rejected to accepted, on their own case or another, over the 527 decidable kernels judged before the revision (three kernels from calls resent after no answer add one own-case rejection, hence the 23 own-case rejections of Section 5.1); none moved the other way. Under the repaired release each frontier model solves three to five more contracts than under the campaign version, and Qwen3.5-397B-A17B one more (Appendix K.6).

| change | found by | contracts | new | recorded kernels |
|---|---|---|---|---|
| revision (App. K.2) | receipts of both populations | 5 | 5 | 29 own-case, 52→23 rejected |
| repair (App. K.3) | reading rejected kernels | 7 | 7 | 22 on other cases |
| repair (App. K.4) | shipping kernels | 16 | 7 | one own-case, 23→22 rejected |
| statement (App. K.5) | shipping kernels | 9 | 2 | none |

in our code tests both conditions against the two trees and every record, so every recorded verdict carries over to the released corpus. All 84 contracts calibrate and all 30 gates pass on the campaign version, the repaired release and the released corpus. The released corpus changes 21 contracts of the campaign version (content digest `b91f9da99dc61909`, path digest `d8bf98b18c04c693`). These include the five revised ones, which the later changes touched again. Rendered from the released corpus, 72 prompts are byte-identical to those of the first round and 12 differ (Appendix N). The code release will include the judging before the revision, the campaign version's records and the repaired ones.

Table 13: The versions of the corpus, by content digest, in order.

| version | content digest | what it is and what used it |
|---|---|---|
| before the revision | `e40070a94e37ceeb` | the corpus the prompts were rendered from; the first judging of the evaluation |
| campaign version | `d2941db429f63999` | after the revision of five contracts; the campaign-version records |
| repaired release | `e8a9e54741a15e26` | the campaign version with the repairs of Appendices K.3 and K.4 (all but the alternative reference of `zero_point_from_minmax`) and with the two accumulation statements; the evaluation's records |
| released corpus | `b91f9da99dc61909` | with that alternative reference and seven division statements; what ships, and to which the repaired release's verdicts carry over |

### K.2 The revision of five contracts

**Discovery of the defects.** Before the revision, over every decidable kernel of both populations, six contracts accepted fewer than half of their kernels. We read their receipts: of the six contracts that accepted fewer than half of their kernels across both populations, five were defective, and we revised them. On the sixth, `fp8_grouped_gemm_moe`, which accepted 6 of 13, every rejection is a kernel that returns FP32 where the prompt states FP16, which we count as the kernel's interface error. A cutoff of this kind finds contracts that are too strict, not contracts that are too permissive, nor defects that reject only some correct kernels, such as the seven of Appendix K.3.

**Four INT4 quantizers.** `int4_per_row_quant`, `int4_per_channel_quant`, `int4_per_group_quant` and `int4_per_tensor_quant` declared their unpacked codes exact while their scale verdict admitted two ULPs, the incoherence of Section 2: the rejected kernels compute $\mathrm{absmax}/7$ as an IEEE quotient, the reference multiplies by the reciprocal and lands one ULP away on some rows, and the kernels' codes are exact at their own scale. The four contracts now declare

membership as their INT8 siblings do, and each rejects all five of its defect mutants and accepts all three of its lawful variants. In the evaluation, 24 of the 25 kernels that the old verdict rejected on these contracts are members at their own scale, and 22 of these on every declared case and seed.

**`llm_int8_linear`.** The task states the dequantization as $\mathrm{INT32} \times \mathrm{row_scale} \times \mathrm{weight_scale}$, and every kernel that any model wrote for it computes that, but the reference divided the INT8 path by an additional $127^2$, so its outputs were $16{,}129$ times smaller than the task describes; its tolerance looked vacuous because it was $51$ times the median magnitude of a wrong reference, and a kernel returning all zeros satisfied it. The reference now computes what the task states. Split-K then moves the output beyond the relative term where the INT8 and FP16 partial products nearly cancel, by up to $1.59$ on Blackwell and $1.88$ on Ada, so the absolute term is now $3.2$, twice the spread on the device where it was set. The old reference rejected all 5 decidable evaluation kernels, and the revised one accepts 5.

**Other contracts.** The revision changes the declarations of five contracts and one reference. Every other contract is byte-identical between the two trees, so judgements of the other 79 carry over, and every kernel written for the five revised contracts was judged again, in both populations, against the campaign version (content digest `d2941db429f63999`, path digest `858b557a560d02a4`; the tree before it was `e40070a94e37ceeb`), except in the cross-device pass of Appendix I.

### K.3 Seven defects that rejected kernels revealed

Under the campaign version 191 kernels of the evaluation were unsolved; we read the source and the receipts of each of the 52 among them that built and were rejected on at least one cell. Where several models failed the same cells, or a kernel implemented the declared rule as written, we wrote variants of the contract's reference that differ from it only where the contract leaves a choice free: the order of a reduction, contraction to fused multiply-adds, or the arithmetic path of the scale. We judged each variant by the campaign version's verdict on every case at the three seeds; a contract that rejects such a variant rejects correct kernels. We then judged every built kernel of each such contract again with the defect removed from outside the contract, the *corrected judge*, and required that the reference and every variant be accepted and that no kernel the campaign version solves be rejected. By this procedure 22 of the 52 rejected kernels are correct and were rejected by defects in seven contracts (Table 14).

**Origins of the defects.** Four of the defects concern a value that the contract fixes only up to rounding. In `w8a8_linear`, the reference's statistics leave the largest element of one row two float32 ULPs short of the midpoint between two fp16 values, and a two-pass block reduction puts it on the midpoint; the row's scale then moves by one fp16 step, and its outputs differ from the reference's by up to $3.2$ against a tolerance of $0.1$ plus five per cent; the three kernels that implement the declared numerics as written return identical outputs and were rejected on the same $146$ elements. The witnesses of `layernorm_quant` and `hadamard_quant_int8` did not admit a one-code move where the float32 reference's value was exactly a tie, although the value and its ties move with the reduction or butterfly order, and `fp8_kv_decode_attention` bounded its output by two fp16 ULPs of a float32 reference, while the rejected outputs are within that bound of a float64 evaluation. The other three are defects of the contract's text or verdicts: `int8_per_row_quant`'s reference floored the scale at $10^{-10}$ while its prompt states $10^{-12}$; `fused_rope_fp8_quant`'s code-step verdict did not admit a rotation contracted to fused multiply-adds, which the CUDA compiler emits by default; and `mla_kv_latent_fp8_quant` judged one of its two code outputs by membership and the other by the code-step verdict it had replaced, so an IEEE quotient $\mathrm{absmax}/448$, which differs by a ULP from the reference's multiplication by the reciprocal, was accepted on one output and rejected on the other.

**The repair.** Each defect is corrected in the contract itself: the floor is applied, `fused_rope_fp8_quant` and `fp8_kv_decode_attention` declare an alternative reference (the rotation with fused multiply-adds, the formula in `float64`), the two witnesses are centered on the declared transform in `float64`, `w8a8_linear` applies its tolerance around every output that the reduction order and the scale's path can produce, `mla_kv_latent_fp8_quant` judges both code outputs by

Table 14: The contracts whose campaign version rejected correct kernels. *Rejected variant*: a variant of the reference that differs from it only where the contract leaves a choice free, and the cells of the campaign version's verdict that reject it. *Kernels*: kernels of the evaluation campaign that the campaign version rejects and the repaired contract solves.

| contract | defect | rejected variant | kernels |
|---|---|---|---|
| `int8_per_row_quant` | the reference floors the scale at $10^{-10}$; contract and prompt declare $10^{-12}$ | the declared rule (3 of 18 cells) | 6 |
| `layernorm_quant` | one-code moves measured from the float32 reference, whose exact ties are held out; the scale within two ULPs of the reference's; zero-row and tie cases reached only in the reference's rounding | a block reduction (6 of 21 cells) | 6 |
| `w8a8_linear` | an output tolerance that is exceeded when the layer norm's reduction order moves a code or a row scale | a block reduction (9 of 18 cells) | 5 |
| `hadamard_quant_int8` | one-code moves measured from the float32 reference, whose exact ties are held out | butterfly stages reversed (4 of 15 cells) | 1 |
| `fused_rope_fp8_quant` | code steps of a rotation contracted to FMA are not lawful moves | the rotation as FMA (5 of 18 cells) | 1 |
| `mla_kv_latent_fp8_quant` | `k_rope` left on the code-step verdict that membership replaced for `c_kv` | IEEE-division scale, reciprocal codes (5 of 15 cells) | 1 |
| `fp8_kv_decode_attention` | a two-ULP bound measured from a float32 reference | none; the rejected outputs are within the bound of a float64 evaluation | 2 |

membership, and the affected zero-row and tie cases are rebuilt. All 30 release gates pass after the repair.

**Judging again.** Every kernel of the evaluation campaign was judged again under the repaired release. The 65 contracts that it leaves unchanged relative to the campaign version reproduce all 5,676 of their cell verdicts. On the repaired contracts, the newly solved kernels are exactly the 22 that the corrected judge solves and the one of the shipping-kernel repair below, and no solved kernel becomes unsolved; the repaired release and the corrected judge agree on 963 of the 969 cells both decide. None of the 112 decidable development kernels of the repaired contracts changes its own-case verdict.

### K.4 The repair the shipping kernels prompted

The first shipping-kernel pass exposed three verdict families that rejected correct outputs, and the second pass a contract whose text admits two readings (Appendix G). The repaired release and the released corpus change 16 contracts for them, 7 of which neither the revision nor the repair above had touched. A rejected correct output is on record for four of those seven (`fp8_gemm_pertensor`, `int8_per_group_quant`, `w8a8_dynamic_gemv_m1` and `zero_point_from_minmax`); the other three (`int8_per_channel_quant`, `int8_per_tensor_quant` and `fp8_per_token_act_quant`) gained the third path with their siblings.

- `fp8_gemm_pertensor` and `fp8_kv_decode_attention` now declare the order-independent bounds of Appendix C, which admit FP8 tensor-core accumulation and half-precision probabilities.
- Twelve quantizer contracts, thirteen witnesses in all, declare the third path of Appendix B. Among them are four of the seven contracts of Appendix K.3 (`hadamard_quant_int8`, `int8_per_row_quant`, `layernorm_quant` and `mla_kv_latent_fp8_quant`) and the four INT4 quantizers of the revision. A record shows the path rejected on two of the twelve, `int8_per_row_quant` and `int8_per_group_quant`, and the others received

it with them. `w8a8_dynamic_gemv_m1` holds its output to two ULPs of the interval that such codes can produce, as `w8a8_linear` does.

- `zero_point_from_minmax` gains an alternative reference that widens a one-sided range to include zero. It equals the reference on every input that keeps the zero-inclusive convention, bit for bit on every other case at three seeds, and it is the only change of this row that reached the released corpus without the repaired release.

One kernel counted as an error of its own under the campaign version is correct under this repair. Claude Sonnet 5's `int8_per_group_quant` multiplies by $127/\mathrm{absmax}$, as vLLM's and SGLang's quantizers do, and neither path the campaign version declares reproduces its codes at ten near-tie positions of its own case. The campaign version rejects that path's variant and this kernel on all $24$ cells, at a few near-tie codes each, and the repaired release accepts both, as it accepts the reference.

### K.5 Precisions stated without a change of verdict

The repaired release states fp32 accumulation for `gptq_linear` and for the fp16 outlier GEMM of `llm_int8_linear`, whose tolerances a half-precision accumulation misses (Appendix G); elsewhere a contract admits what its own bound or tolerance allows, so the corpus has no single accumulation policy. The released corpus also states that seven quantizers that return a `float32` scale divide in `float32`, which torchao's affine quantizer does not (Appendix G). None of these statements is read by a verdict, so no kernel's verdict changes with them, though a statement in a prompt can change what a model writes (Appendix N); each of the 12 evaluation kernels of the two accumulating contracts that the campaign version decides is solved under both the campaign version and the repaired release.

### K.6 What the changes did to the results

**Solved kernels.** Under the repaired release each frontier model solves three to five more contracts than under the campaign version, so that their range moves from 67–76 to 71–80, and Qwen3.5-397B-A17B solves one more. The 22 kernels of the defect repair all passed their own case, and were 22 of the 28 kernels that passed their own case and failed another, which leaves the 6 wrong ones of Section 5.1. The kernel of the shipping-kernel repair did not pass its own case under the campaign version.

**The rule comparison.** Before the revision, the contract rejected 52 of the evaluation's decidable kernels, KernelBench-X accepted 30 that the contract rejected, and KernelBench's `allclose` rejected 28 that it accepted. On the same 527 kernels, which exclude the three obtained by resending unanswered calls, these counts are 22, 5 and 58 under the repaired release; the revision alone brings the first to 23, and the shipping-kernel repair the rest of the way. All but 5 of KernelBench-X's 30 apparent wrongful acceptances were therefore wrongful rejections by our contracts. Table 1 and Table 2 also count the three kernels obtained later, one of which the contract rejects and KernelBench-X accepts, hence their 23 and 6. The kernel of the shipping-kernel repair moves every rule's row of Table 2 by one: under the campaign version the calibrated tolerance, KernelBench-X and FlashInfer-Bench at $0.95$ accepted it on its own case where the contract rejected it, and under the repaired release `allclose` and FlashInfer-Bench at its default ratio reject it where the contract accepts it. In the every-case comparison of Appendix F, the 22 had counted as wrongful acceptances for the rules that accepted them on every cell.

**Limits of the search.** Every change was found by reading rejections, and no change turns an acceptance into a rejection. For any rule this can only raise its wrongful rejections and lower its wrongful acceptances, and a contract that accepts a wrong kernel is tested only by the mutants of Appendix D. A comparison of rules depends on the ground truth it is scored against, and here that ground truth changed after the evaluation had been judged; the counts that hold under every version are in Section 5.2. The second round (Appendix N) tests the frozen contracts on new kernels; it can find a contract that rejects a correct kernel, not one that accepts a wrong kernel.

## L THE DEVELOPMENT POPULATION

Before the evaluation campaign we generated kernels in two sets. The *development campaign* ran eight models (Qwen3.8-Max, DeepSeek-V4-Pro, GLM-5.3, Kimi-K3, and four sizes of Qwen3.5) for three rounds of one attempt per contract each on 43 contracts. Of its 1,032 attempts, 638 built, 393 did not, and one returned no kernel file; 586 of the built kernels were decidable on the contract's own case, and 519 were accepted. The *archived run* is nine single-shot runs on the other 41 contracts, one by each of the same eight models, made between the development campaign's first and second rounds on 25 of them, and one by DeepSeek-V4-Flash a month earlier on 39 of them. Of its 237 kernels, 130 were decidable, 115 accepted and 15 rejected. Together they form the 716 decidable kernels over 82 contracts (two contracts have no decidable kernel) that this paper calls the development population. We developed and validated the instrument on it, so its columns in Table 2 are not an independent replication, and we do not rank models on it.

**Reason for the re-run.** Of the 1,032 development attempts, 393 did not build, and 137 of these stem from the prompt, which said nothing about which headers the harness supplies, which helper macros exist, which build flags are in force or which return types the binding accepts. With a build-environment section added to the prompt, one model (GLM-5.3) on the same contracts went from 23 compile failures over its three rounds of 43 contracts to 2 in one round, and the evaluation campaign uses that prompt.

**Prompt contents.** Every one of the 1,032 prompts sent in the development campaign carries a declared-numerics section listing every clause its contract fixes, an acceptance section, the argument names of its signature and the source of its generator, and 0 fail that audit. None of the 1,909 prompts saved from all development runs, a superset of those attempts, contains the reference, and no attempt had a tool that reads files or runs a shell; 997 attempts are a single turn, and the 45 tool calls in the rest were refused.

**The development campaign on every declared case.** Re-judging its 586 decidable kernels on every declared case at three seeds (7,152 cells) reproduces the single-case verdict on every kernel and shows that 23 of the 519 single-case acceptances, 4.4%, fail another case, while none changes in the other direction. Of those, 10 disappear under the released corpus, 7 on `hp_gemv_bf16` under the order-independent bound it declares and three under the repaired contracts, whose development kernels we judged again on every case, and 13 remain attributable to the kernel.

## M WHAT IS NOT FINISHED

- **Two single-shot rounds per model.** Per model the solved count moves by at most 7 between the rounds, but which contracts are solved moves more (Appendix N), and in each round some calls that spent the whole budget on reasoning received a second draw (Appendices E and N); each table is a measurement under one draw, not a ranking.
- **No outside adjudication.** The kernels on which the rules and the contract disagree were adjudicated by the authors.
- **Model identity.** Each model is named by the identifier the gateway serves (Appendix E).
- **Ground truth changed after judging.** We changed 21 contracts after the evaluation was judged (Appendix K). Each change corrects a reference to the stated task, admits arithmetic that the contract's text leaves free or that shipping kernels perform, or states a precision without changing a verdict, and the earlier records will be released; but a repair made after reading the kernels could be fitted to them, and the search that found the defects cannot find a contract that accepts a wrong kernel; a second round generated after the last change found no correct kernel that the frozen contracts reject, except possibly two on the contracts named for FP8 over INT8 data (Appendix N).
- **Ten loose tolerances.** They are measured, and on their own cases the accepted kernels of both populations use under half of them but for one `w8a8_linear` kernel (Appendix J); they are not tightened, so that no recorded verdict changes.

- **No single accumulation policy.** Which accumulation a contract admits follows from its own bound or tolerance (Appendices G and K.5).
- **Four repairs to the task text and one hint.** The prompts of 4 contracts omit a zero guard; two contracts are named for FP8 although their tensors are INT8; the prompt's table does not name the header that `CUDART_INF_F` needs; and the prompt of `int4_per_row_quant` still describes a one-code step at a tie as a convention, while its rounding clause and its verdict require ties to even. No evaluation kernel of `int4_per_row_quant` differs from the reference at an exact tie, so that sentence affected no verdict. The hint would say that `data_ptr<__half>()` does not link, a model error behind several build failures (Appendix E). Each is one line in the next corpus version and is not applied here, so that the prompts the released corpus renders differ from those the campaign's models saw only on the changed contracts.
- **Shipping kernels that do not run here.** vLLM's Machete and DeepGEMM's Hopper and data-center Blackwell kernels need GPUs we do not have, and DeepGEMM's path for `sm_120`, which the GB10 runs, takes only power-of-two scales. TensorRT-LLM 1.2.0 is judged only on the Ada card, with its attention operators, which need its runtime, left out.
- **The calibrated tolerance under other choices.** Its counts with a floor of one step of the output dtype are measured only on the contracts' own cases, and its counts with INT4 codes unpacked only from a re-run of those kernels on the Ada card (Appendix F).
- **Trace-derived cases not merged.** Trace-derived workload cases pass every gate but are not in the released corpus, because adding them would change the corpus digest and require re-judging every population.

## N A HELD-OUT ROUND

**Design.** The changes of Table 12 were made after reading the first round's kernels. To test the frozen contracts on kernels that no change was fitted to, we asked every model once more on each of the 84 contracts after the released corpus was frozen. The prompts were rendered from the released corpus by the first round's renderer and template; 72 of them are byte-identical to the first round's, and the 12 that differ belong to contracts changed after judging. The client, the gateway, the model identifiers and the output budget are those of Appendix E, and the first round's retry rule was applied to every model alike: a call that spent the whole budget without code was repeated once with 120,000 tokens (52 calls, from Kimi-K3, DeepSeek-V4-Pro and GLM-5.3), and a call without an answer was sent once more (6, from Qwen3.8-Max). Every kernel was judged as the first round's were, on the RTX PRO 6000, on every declared case at three seeds, by the released contract and the five rule settings. No contract, prompt or harness changed after the round began.

Table 15: The held-out second round beside the first, per model: kernels that build, kernels the contract rejects on their own case, and contracts solved (every declared case at three seeds), each as first round / second round. *Moved*: contracts solved in one round and not the other, on prompts identical in both rounds / on the 12 prompts that changed.

| | built | | rejected | | solved | | |
|---|---|---|---|---|---|---|---|
| model | R1 | R2 | R1 | R2 | R1 | R2 | moved |
| Claude Sonnet 5 | 80 | 80 | 4 | 6 | 72 | 71 | 10/5 |
| GPT-5.6-sol | 79 | 80 | 1 | 1 | 77 | 76 | 2/3 |
| Kimi-K3 | 82 | 79 | 2 | 2 | 80 | 76 | 6/4 |
| DeepSeek-V4-Pro | 71 | 71 | 0 | 2 | 71 | 69 | 18/4 |
| GLM-5.3 | 80 | 74 | 1 | 1 | 79 | 72 | 14/3 |
| Qwen3.8-Max | 80 | 80 | 1 | 0 | 79 | 80 | 4/1 |
| Qwen3.5-397B-A17B | 52 | 49 | 12 | 8 | 39 | 41 | 18/4 |
| Qwen3.5-35B-A3B | 6 | 11 | 2 | 7 | 4 | 4 | 4/0 |

**Per model.** Per model, the solved count moves by at most 7 between the rounds (Table 15), but which contracts are solved moves more: over the models, 24 of the 96 model–contract pairs on the

12 changed prompts change status, against 76 of the 576 on the byte-identical ones. `int8_asym_per_row_quant`, whose prompt gained a sentence on the precision of its division while its range clause is unchanged, is solved by every frontier model in the first round and by one of them in the second. Across the two rounds, all 6 frontier models solve 33 contracts every time. A single round measures conformance to stated numerics and does not rank models of similar strength. Claude Sonnet 5 again decodes FP4 code 8 as $+0.0$, at the same 16,333 positions.

**Every contract rejection read.** Of the round's built kernels, 35 are not solved, and we read the source and the receipt of each: 6 round ties away from zero with `roundf` where the contract rounds half to even, 4 apply the zero guard to the absmax instead of the scale, 2 do not widen an asymmetric quantizer's range to include zero, 1 decodes the negative-zero code as $+0.0$, 3 cannot run a declared case (a single group size, a row count divisible by 16, a launch that fails on one shape), 3 return an output of the wrong bit width or size (2 of them FP32 where the prompt states FP16, which we count as errors, as in the first round, Appendix E), and 14 are wrong on ordinary inputs. The other 2 miss the tolerance almost everywhere on the two contracts whose name and prose say FP8 while their tensors are INT8 (Appendix J), where the task statement may have misled the model; we count those neither as errors nor as correct. The 33 others break a clause their prompt states, so the frozen contracts reject none of them wrongly.

**Repeats of the first round.** The round can find a contract that rejects a correct kernel, not one that accepts a wrong kernel, and most of it repeats the first: 433 of its 524 decidable kernels return the same output on the own case as the same model's first-round kernel. On the 91 that return a new output or had no first-round verdict, the contract rejects 20, all among the kernels read above, while `allclose` rejects 11 correct kernels and KernelBench-X accepts 6 wrong ones.

**The rules on the round's kernels.** Pooled over the models as test cases for the rules, the round has 524 decidable kernels. Table 16 compares each rule setting with the contract on them, counted as in Table 2. Both kinds of error recur: the strict settings reject correct kernels and the loose settings accept wrong ones. No single threshold separates the two either: on the 513 kernels left after the exclusions of Figure 2 (one more has no finite distance on its first output), the best threshold on relative $L_1$ misclassifies 9 and the best on the fraction of differing elements 12, against 23 for a rule that accepts every kernel.

Table 16: The five rule settings against the contract on the held-out round's decidable kernels, on the own case at the visible seed, counted as in Table 2.

| rule | accepts wrongly | rejects wrongly |
|---|---|---|
| FlashInfer-Bench, ratio 0.95 | 10 | 5 |
| KernelBench-X | 8 | 1 |
| calibrated tolerance | 8 | 64 |
| KernelBench `allclose` | 2 | 53 |
| FlashInfer-Bench, ratio 1.0 | 3 | 54 |