

Rethinking Correctness Evaluation for GPU Kernel Optimization*

Zirui Wang

The Chinese University of Hong Kong
China

zrwang@cse.cuhk.edu.hk

Yunbo Ni

The Chinese University of Hong Kong
China

ybni@cse.cuhk.edu.hk

Shaohua Li

The Chinese University of Hong Kong
China

shaohuali@cse.cuhk.edu.hk

Abstract

LLM-based kernel generation systems primarily judge correctness through tolerance-based numerical comparison: an elementwise-close output passes. This check is necessary for floating-point software but does not ensure safe substitution inside a full model. We study this gap through a substitution-oriented stress test: for each kernel site, we apply dtype-scale bounded output perturbations while fixing weights, inputs, and remaining computation, then measure downstream task-output changes. Across 13 models, 9,510 runs, and 8.62M evaluated cases, these within-tolerance perturbations induce 371,537 task-output flips (4.3%). Sensitivity is concentrated at attention probability, activation, and normalization sites; lower-precision configurations with larger perturbation budgets show up to 50× higher flip rates; output/scoring sites remain near zero. Tensor distance alone does not explain this pattern; operator role provides additional context. An actual CUDA GELU substitution in ViT passes every local check (max error 4.74×10^{-4}), yet accumulated drift exceeds tolerance at 11/12 layers (max error 1.6947), changing top-1 predictions. These findings motivate extending kernel evaluation beyond local tolerance to also consider operator-level behavioral risk, precision sensitivity, and composition safety.

1 Introduction

LLM-based systems are increasingly used to generate and optimize low-level GPU kernels. Benchmarks and agentic workflows, including KernelBench, CUDA-L1, KernelAgent, CUDA Agent, CudaForge, and KernelBlaster, seek kernels that are both *correct* and *fast* [3–5, 7, 8, 14]. Their growing capability makes the reliability of the correctness oracle increasingly important.

In practice, kernel correctness is operationalized as finite testing plus tolerance-based numerical comparison. An allclose-style predicate accepts a candidate when each element differs from the reference within absolute and relative tolerances [11]. This is necessary because mathematically equivalent floating-point computations can differ across operation orderings, platforms, precision modes, and backends [9, 13]. PyTorch explicitly warns that complete reproducibility is not guaranteed across releases and platforms [10].

However, a local tolerance pass is weaker than a valid replacement claim. It confirms tensor closeness under a numerical budget, but not that substitution preserves downstream behavior, respects the precision policy, composes safely, or improves performance. Recent work has begun to expose this gap: robust-kbench shows exploitable loopholes in existing benchmark oracles [6]; the Correctness Illusion shows that tolerance-based oracles can overestimate correctness under stronger fuzzing [12]; ProofWright reflects the concern that runtime testing alone is insufficient for CUDA correctness [1]. Our work focuses on a complementary question: even when local numerical deviations are bounded, how risky is the kernel as a substitute inside an end-to-end model?

We perturb one kernel-wrapper output at a time by a bounded constant, keep everything else fixed, and measure downstream task-output changes. Across 13 models and multiple task families, bounded local changes induce substantial task-output flips, concentrated around attention, activation, and normalization sites, with higher rates under lower precision and larger perturbation budgets. This is consistent with the expectation in compiler and tensor-program optimization that transformations should preserve semantics [2, 15], and motivates treating tolerance as one necessary layer of evidence rather than the sole correctness signal. A stronger evaluation should also report behavioral sensitivity, operator context, precision policy, and performance value under the same replacement setting [6–9, 12].

2 Why Tolerance Is Not Enough

A tolerance-based oracle is a local, per-element predicate; it does not establish safe replacement inside a model. For tensors x and y , an allclose-style check accepts when

$$|x_i - y_i| \leq \text{atol} + \text{rtol} \cdot |y_i|$$

for every element i [11]: atol permits a fixed absolute gap, and rtol permits an additional gap proportional to $|y_i|$. This predicate accommodates floating-point implementation differences, but is insufficient as the sole evidence of a meaningful optimization.

A simple thought experiment shows why passing tolerance does not entail valid optimization. Suppose a candidate calls the reference and adds a bounded output offset. It may pass without algorithmic improvement, yet consume

*Published as an extended abstract at the 2nd International Workshop on Language Models and Programming Languages (LMPL 2026).

the error budget and push downstream decisions past task-relevant thresholds. In isolation the offset is a local numerical difference; inside a model it becomes a substitution whose effect depends on operator position, invocation frequency, precision policy, and decision margins. Correctness evaluation should therefore distinguish *bounded observational similarity*, where the outputs are elementwise close, from *valid replacement*, where the substitution preserves intended model behavior.

A separate observation illustrates a related gap. CUDA-L1 submissions are intended to implement optimized GPU kernels, yet some pass correctness by never running their optimized code at all. In a local audit of 404 Level-1 artifacts, 120 passing submissions returned an unoptimized fallback output rather than the intended kernel; 13 borderline cases were excluded. Because CUDA-L1 accepts a speed pass whenever any one of five reference baselines is exceeded [7], a fallback path can still clear both correctness and performance gates by outperforming the weakest reference. This does not invalidate CUDA-L1’s design, but it shows that passing tolerance plus a speed ratio does not guarantee the submission actually executes the optimization it claims.

3 Preliminary Study Design

The perturbation study tests whether bounded, dtype-scale changes to one kernel output preserve downstream task outputs. For each kernel wrapper in a model, we run one clean baseline and five perturbation runs per dtype $\times$ benchmark $\times$ kernel tuple. Let f_k denote the kernel wrapper at site k . The clean path computes $z = f_k(x)$ and passes z to downstream layers; the perturbed path passes $\tilde{z} = z + \epsilon$, where ϵ is a signed scalar added to every element:

$$\tilde{z}_i = z_i + \epsilon, \quad |\epsilon| \leq \text{atol}$$

Weights, inputs, and remaining execution stay fixed. This probes the tolerance criterion rather than modeling a particular optimized kernel. The five ϵ values per dtype are drawn from the corresponding precision bucket (bf16/fp16: $|\epsilon| \in [10^{-3}, 10^{-2}]$; fp32: $|\epsilon| \in [10^{-5}, 10^{-4}]$). For analysis, we group sites by the operation indicated by wrapper names: e.g., `attention_softmax` as attention probability, `gelu` as activation, and `rmsnorm` as normalization. These coarse labels describe model roles, not CUDA primitive classes; ambiguous or fused wrappers may be misclassified.

A *task-output flip* is a change from the clean prediction (label, answer, or transcription), not necessarily an error. Flip rates count changes over evaluated cases, including repeated inputs across runs; task quality is assessed separately by accuracy (Acc), exact match (EM), or word error rate (WER).

A second experiment tests whether repeated use of a locally tolerance-passing GELU changes predictions. We replace PyTorch GELU with a KernelBench rank-01 tanh-form CUDA GELU across all 12 encoder layers and trace 13 ImageNet images whose top-1 predictions differ between

Table 1. Study design summary.

Component	Setting
Scope	13 models over vision cls., CLIP cls., extractive QA, NLI, MMLU, CTC ASR, and seq2seq ASR.
Perturbation	Add a constant to one wrapper output; keep weights, inputs, and other execution fixed.
Precision budget	Five signed constants per dtype bucket: bf16/fp16 from 10^{-2} – 10^{-3} , fp32 from 10^{-4} – 10^{-5} .
Formal grid	One clean baseline + five perturbation runs per dtype $\times$ benchmark $\times$ kernel tuple.
Role grouping	Group wrapper names into coarse operator roles for analysis.

paths. At each encoder layer, we stack all 13 images and run `allclose` with `atol = rtol =  $10^{-2}$` , comparing local same-input GELU outputs against accumulated path drift (12 check rows).

4 Preliminary Findings

4.1 Finding 1: within-tolerance perturbations flip task outputs across all 13 models.

Even dtype-scale perturbations within standard tolerance budgets substantially alter downstream task outputs. Across 9,510 completed perturbation runs covering 8.62M evaluated cases, we observe 371,537 task-output flips (4.3% overall). This pooled rate measures output disagreement, not accuracy loss or comparable severity across tasks.

Table 2 shows nonzero but sharply varying flip rates across all models. Wav2Vec2 reaches 33.31% aggregate (49.03% in bf16 alone), while DeBERTa-NLI stays at 1.32%, a 25 $\times$ range. Table 3 lists each model’s most sensitive kernel, where sensitivity can be near-total. In bf16, Wav2Vec2 has 99.53% flips with a 1414.0% WER increase; DeBERTa-QA has 94.30% flips with a 94.5% EM decrease; ViT-Base has 78.42% flips with a 70.2% accuracy decrease. These selected high-sensitivity settings demonstrate task-metric degradation as well as output disagreement; they do not estimate average quality loss across all settings.

4.2 Finding 2: sensitivity varies with operator role and precision configuration.

Behavioral sensitivity is confined to a few operator roles. Figure 1 shows that attention-probability sites reach maximum observed flip rates of 100% in ViT-Base, ViT-Large, and DeBERTa-QA. Activation sites reach 100% in Wav2Vec2 and 81.6% in Qwen3-8B; normalization reaches 100% in Wav2Vec2 and 78.2% in Gemma-12B. By contrast, output/scoring kernels never exceed 0.2% across all 13 models, and embedding sites stay below 4.7%. This pattern is specific to the tested constant offsets; it does not establish an intrinsic sensitivity ranking for arbitrary kernel errors.

Under the chosen budgets, lower-precision configurations show higher rates. Average attention-probability flip rates

Table 2. Perturbation sensitivity by model and dtype. Each row aggregates all perturbation tuples for one model. *Kernel sites*: distinct perturbed kernel wrappers. Flip rates are per dtype; Agg. aggregates across all three.

Type	Model	Task	Kernel sites	Cases/dtype	fp32 flip%	bf16 flip%	fp16 flip%	Agg. flip%
Speech	Wav2Vec2	CTC ASR	21	54k	7.94	49.03	42.96	33.31
Speech	Whisper	Seq2seq ASR	19	49k	0.05	4.20	3.08	2.44
Decoder LM	Gemma-12B	MCQA	23	115k	0.15	10.27	9.97	6.80
Decoder LM	Gemma-E4B	MCQA	23	115k	0.06	9.55	8.49	6.03
Decoder LM	Qwen3-8B	MCQA	19	95k	0.14	9.93	9.72	6.60
Decoder LM	Qwen3-14B	MCQA	19	95k	0.09	6.53	6.28	4.30
Vision	ViT-Base	Image cls.	16	320k	0.12	7.52	6.94	4.86
Vision	ViT-Large	Image cls.	16	320k	0.06	6.48	6.03	4.19
Vision	CLIP-ViT	Image cls.	14	280k	0.07	7.16	6.59	4.61
Vision	ConvNeXt	Image cls.	15	300k	0.04	3.35	3.17	2.19
Vision	DINOv2	Image cls.	15	300k	0.02	3.43	2.91	2.12
Encoder LM	DeBERTa-QA	Extractive QA	21	420k	0.18	7.24	6.74	4.72
Encoder LM	DeBERTa-NLI	NLI	21	420k	0.01	2.00	1.96	1.32

Table 3. Most sensitive kernel per model. Highest aggregate flip rate over kernels and dtype $\times$ benchmark settings, pooling the five perturbation runs. *Metric Δ* : relative change from clean baseline; positive WER Δ = worse, negative Acc/EM Δ = worse.

Model	Dtype	Benchmark	Metric	Kernel	Flip%	Metric Δ
Wav2Vec2	bf16	LibriSpeech test-clean	WER	feature_activation	99.53	+1414.0
Whisper	fp16	LibriSpeech test-other	WER	gelu	18.20	+2.3
Gemma-12B	fp16	MMLU	Acc	rmsnorm	66.34	-56.9
Gemma-E4B	bf16	MMLU	Acc	rmsnorm	60.82	-49.1
Qwen3-8B	fp16	MMLU	Acc	mlp_activation	75.56	-66.2
Qwen3-14B	fp16	MMLU	Acc	mlp_activation	45.16	-40.9
ViT-Base	bf16	ImageNet-A	Acc	attention_softmax	78.42	-70.2
ViT-Large	bf16	ImageNet-A	Acc	attention_softmax	69.86	-62.6
CLIP-ViT	bf16	ImageNet-Sketch	Acc	quick_gelu	28.86	-6.5
ConvNeXt	bf16	ImageNet-A	Acc	gelu	27.04	-13.3
DINOv2	bf16	ImageNet-A	Acc	gelu	23.32	-15.6
DeBERTa-QA	bf16	SQuAD v1	EM	attention_softmax	94.30	-94.5
DeBERTa-NLI	fp16	ANLI R1-R3	Acc	attention_softmax	38.70	-26.7

rise from 1.3% in float32 to 65.5% in bfloat16, roughly 50 $\times$ (Figure 2). Activation follows the same pattern, climbing from 0.7% to 18.3% (25 $\times$); normalization from 0.3% to 6.3% (21 $\times$). Output/scoring, however, remains at 0.0% across all three precisions. Because perturbation ranges also differ by dtype, these comparisons do not isolate the effect of precision alone. They motivate reporting precision and error budgets together.

4.3 Finding 3: tensor distance alone does not explain output changes.

Output mean absolute error (MAE) averages elementwise differences between perturbed and reference outputs. Figure 3 shows no simple monotonic relation with flip rate: similar MAEs span 0–60% flips, and many high-MAE tuples produce none. The color grouping by operator role reveals the structure behind this dispersion: attention-probability and activation points concentrate at the top of the plot, while output/scoring points remain at 0% regardless of their MAE. For example, in CLIP-ViT bfloat16 on ImageNet-A, feature L2 normalization has higher MAE than quick_gelu (2.04 vs. 0.81) but fewer flips (14.3% vs. 28.5%). Tensor distance should therefore be interpreted alongside operator context; this exploratory comparison does not establish their relative predictive power.

4.4 Finding 4: locally tolerance-passing kernels can change predictions under repeated substitution.

The ViT GELU trace shows that a kernel passing every local check can still change predictions after repeated application. As Table 4 shows, comparing optimized and reference GELU outputs on the same inputs yields 0/12 layers violating the tolerance, with a maximum absolute error of only 4.74×10^{-4} , well within the CUDA-L1 tolerance ($\text{atol} = \text{rtol} = 10^{-2}$). The unmodified path and reference wrapper produce identical outputs on these traces. Once the optimized GELU is applied repeatedly through 12 ViT layers, however, small per-layer differences accumulate: inputs to later GELUs diverge between the reference and optimized paths, causing 11/12 layers to exceed $\text{atol} = \text{rtol} = 10^{-2}$ on both accumulated input drift and accumulated output drift, with a maximum absolute error of 1.6947. Local tolerance checks therefore verify each invocation in isolation but do not capture the cumulative effect of substituting a kernel across an entire model.

5 Implications, Limitations, and Next Steps

These findings motivate substitution-aware correctness for kernel benchmarks. The perturbation study shows substantial output disagreement concentrated in a few operator roles, with higher rates under lower precision and larger

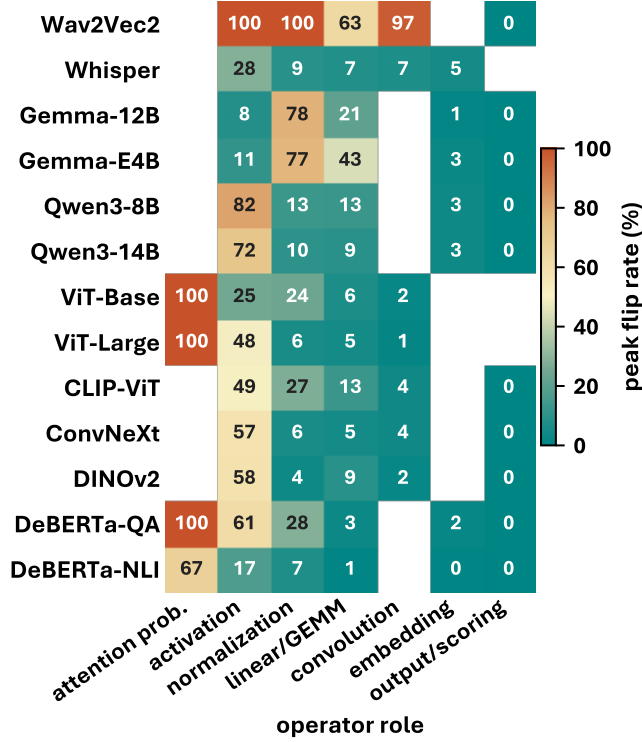


Figure 1. Maximum observed flip rate by operator role and model. Each cell takes the maximum single-run rate over kernels, dtypes, benchmarks, and tested offsets for that model and role; this is not a theoretical worst case.

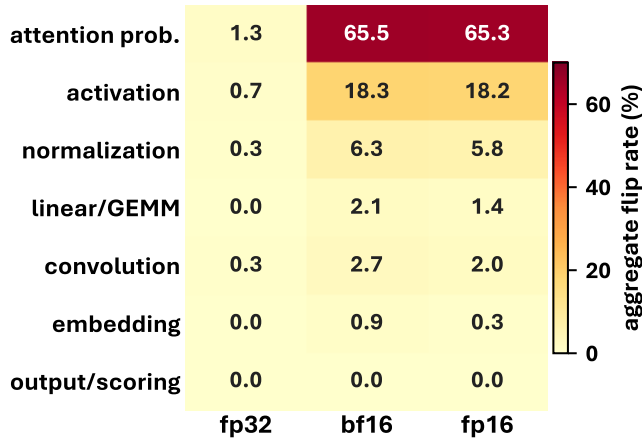


Figure 2. Flip rate by operator role and dtype. Each cell: average flip rate across all models for a given role $\times$ dtype pair.

error budgets; tensor distance alone does not explain the pattern. The GELU trace further shows that a locally passing kernel can change predictions after repeated application. Evaluation should therefore report, alongside local tolerance pass/fail, the operator role and its sensitivity profile, the declared precision policy, accumulated path drift under

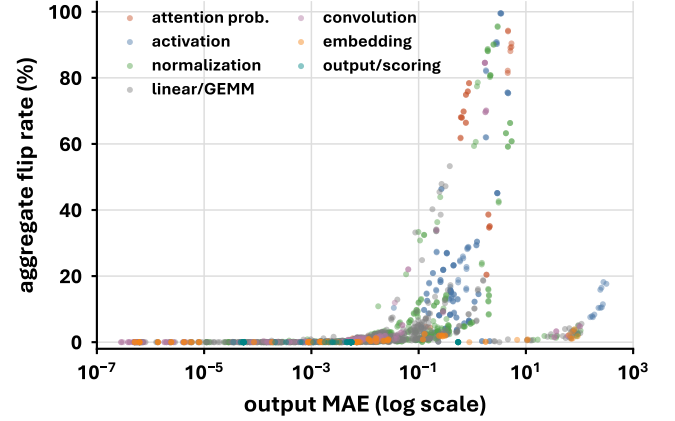


Figure 3. Tensor distance versus behavioral risk. Each point is one model $\times$ dtype $\times$ benchmark $\times$ kernel tuple; x-axis: output MAE; y-axis: aggregate flip rate.

Table 4. Local versus accumulated GELU deviations in ViT-Base/16. 13 flip-selected images; 12 rows = one encoder layer each (13 images stacked); 13/13 top-1 flips reflect selection, not a new rate; tolerance atol = rtol = 10^{-2} .

Signal	Count	Value
Local tolerance violations	0/12	max abs 4.74×10^{-4}
Current vs. ref mismatch	0/12	max abs 0
Input drift violations	11/12	max abs 1.6947
Output drift violations	11/12	max abs 1.6947
Selected top-1 flips	13/13	ref = current, opt differs

repeated application, and whether the performance gain justifies the behavioral cost. Tolerance-based testing remains one component of a stronger claim: the candidate is close locally, stable in context, and worth substituting.

The current evidence has clear scope limits. Constant offsets stress-test tolerance, not real-kernel error distributions. They may violate constraints such as softmax normalization and accentuate role-specific sensitivity. The GELU trace covers one ViT model and 13 selected images rather than a full replacement sweep. All experiments use benchmark slices. Decoder-LM results concern multiple-choice answers; ASR does not establish open-ended generation robustness. We do not compare kernel-induced changes with sampling variability or assess multi-token generation quality. These preliminary results identify an evaluation gap, not a general kernel failure rate. Extending the composition analysis to more models, calibrating error budgets by operator role, precision policy, and replacement density, and evaluating real-world optimized kernel substitutions end-to-end are the immediate next steps.

References

- [1] Bodhisatwa Chatterjee, Drew Zagieboylo, Sana Damani, Siva Hari, and Christos Kozyrakis. 2025. ProofWright: Towards Agentic Formal Verification of CUDA. arXiv:2511.12294 [cs.SE] <https://arxiv.org/abs/2511.12294>

- [2] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, 578–594. <https://www.usenix.org/conference/osdi18/presentation/chen>
- [3] Kaiming Cheng, Laura Wang, Jack Khoo, Mark Saroufim, Wenyan Chi, Jiannan Wang, and Joe Isaacson. 2026. KernelAgent: Hardware-Guided GPU Kernel Optimization via Multi-Agent Orchestration. <https://pytorch.org/blog/kernelagent-hardware-guided-gpu-kernel-optimization-via-multi-agent-orchestration/>. Accessed 2026-06-26.
- [4] Weinan Dai, Hanlin Wu, Qiying Yu, Huan-an Gao, Jiahao Li, Chengquan Jiang, Weiqiang Lou, Yufan Song, Hongli Yu, Jiaze Chen, Wei-Ying Ma, Ya-Qin Zhang, Jingjing Liu, Mingxuan Wang, Xin Liu, and Hao Zhou. 2026. CUDA Agent: Large-Scale Agentic RL for High-Performance CUDA Kernel Generation. arXiv:2602.24286 [cs.LG] <https://arxiv.org/abs/2602.24286>
- [5] Kris Shengjun Dong, Sahil Modi, Dima Nikiforov, Sana Damani, Edward Lin, Siva Kumar Sastry Hari, and Christos Kozyrakis. 2026. KernelBlaster: Continual Cross-Task CUDA Optimization via Memory-Augmented In-Context Reinforcement Learning. arXiv:2602.14293 [cs.LG] <https://arxiv.org/abs/2602.14293>
- [6] Robert Tjarko Lange, Qi Sun, Aaditya Prasad, Maxence Faldor, Yujin Tang, and David Ha. 2025. Towards Robust Agentic CUDA Kernel Benchmarking, Verification, and Optimization. arXiv:2509.14279 [cs.SE] <https://arxiv.org/abs/2509.14279>
- [7] Xiaoya Li, Xiaofei Sun, Albert Wang, Jiwei Li, and Chris Shum. 2025. CUDA-L1: Improving CUDA Optimization via Contrastive Reinforcement Learning. arXiv:2507.14111 [cs.AI] <https://arxiv.org/abs/2507.14111>
- [8] Anne Ouyang, Simon Guo, Simran Arora, Alex L. Zhang, William Hu, Christopher Ré, and Azalia Mirhoseini. 2025. KernelBench: Can LLMs Write Efficient GPU Kernels? arXiv:2502.10517 [cs.LG] <https://arxiv.org/abs/2502.10517>
- [9] PyTorch Contributors. 2026. Numerical Accuracy. https://docs.pytorch.org/docs/2.12/notes/numerical_accuracy.html. Accessed 2026-06-26.
- [10] PyTorch Contributors. 2026. Reproducibility. <https://docs.pytorch.org/docs/2.12/notes/randomness.html>. Accessed 2026-06-26.
- [11] PyTorch Contributors. 2026. torch.allclose. <https://docs.pytorch.org/docs/2.12/generated/torch.allclose.html>. Accessed 2026-06-26.
- [12] Dipankar Sarkar. 2026. The Correctness Illusion in LLM-Generated GPU Kernels. arXiv:2606.20128 [cs.SE] <https://arxiv.org/abs/2606.20128>
- [13] Nathan Whitehead and Alex Fit-Florea. 2011. Floating Point and IEEE 754 Compliance for NVIDIA GPUs. <https://docs.nvidia.com/cuda/floating-point/index.html>. NVIDIA white paper; accessed 2026-06-26.
- [14] Zijian Zhang, Rong Wang, Shiyang Li, Yuebo Luo, Mingyi Hong, and Caiwen Ding. 2025. CudaForge: An Agent Framework with Hardware Feedback for CUDA Kernel Optimization. arXiv:2511.01884 [cs.LG] <https://arxiv.org/abs/2511.01884>
- [15] Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, Joseph E. Gonzalez, and Ion Stoica. 2020. Ansor: Generating High-Performance Tensor Programs for Deep Learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 863–879. <https://www.usenix.org/conference/osdi20/presentation/zheng>