METAKERNELBENCH: MEASURING GPU KERNEL KNOWLEDGE TRANSFER BEYOND CODE

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent GPU kernel optimization agents retain what they learn in knowledge bases or as distilled skills. Kernel benchmarks score each attempt's implementation for correctness and speed but leave the reuse value of retained experience unmeasured. We introduce MetaKernelBench, which measures whether experience distilled from an attempt in one kernel domain-specific language (DSL) improves a fresh attempt at the same problem in another. Its 74 problems are fused subgraphs in six families, each posed as a pair of CuTe DSL and TIRx variants that differ only in the DSL. The agent first attempts each variant solo and is instructed to distill what it learns into a natural-language skill, which is transferred whether or not the source attempt passes verification. The skill is the only extra input to a skill-conditioned attempt by the same model in the other DSL. We compare each skill-conditioned attempt with the solo attempt on the same variant under matched per-attempt budgets, scoring correctness and end-to-end runtime. Across six models and both directions, paired lift over solo attempts ranges from -19% to $+29\%$. Four models gain in both directions, yet regressions occur on 16% to 45% of problems in every model and direction. Outcomes follow the source attempt's result relative to the target's solo attempt rather than source success alone, improving in 71% of comparisons when the source stands above and regressing in 54% when it stands below. MetaKernelBench complements implementation-quality metrics by measuring same-problem cross-DSL kernel knowledge transfer.

1 INTRODUCTION

GPU kernel optimization agents iteratively generate, profile, and revise implementations using execution feedback (Wang et al., 2025; Wei et al., 2025). Recent agents retain what they learn in persistent knowledge bases or distill successful rollouts into skills (Dong et al., 2026; Fu et al., 2026). Kernel benchmarks score the implementation each attempt produces for correctness and speed relative to a reference (Ouyang et al., 2025). Newer benchmarks extend this scoring to other hardware platforms and kernel domain-specific languages (DSLs) and to unseen input configurations (Wen et al., 2026; Younesian et al., 2026). Such per-attempt scores leave the reuse value of retained experience unmeasured. We ask whether experience distilled from an attempt in one DSL can improve a fresh attempt at the same problem in another.

We use *kernel knowledge* to denote problem-, hardware-, and strategy-level principles that guide implementation decisions and can be expressed independently of a DSL's syntax and APIs. Examples include avoiding the materialization of intermediate tensors, overlapping data movement with computation, and dividing work between eager PyTorch and DSL kernels. A *skill* records an agent's findings in natural language, without guaranteeing that they are correct or useful to a later attempt.

Pairing the same problem across DSLs preserves the computational and hardware conditions that make prior experience relevant while changing how the kernel must be expressed. Repeating the problem in the same DSL would let DSL-specific recipes carry over unchanged, whereas switching problems would change which findings remain relevant. Advice about eager PyTorch may still apply regardless of the DSL, but kernel strategies must be re-implemented in the other DSL. This setting does not test generalization to new problems, workloads, or hardware.

We introduce **MetaKernelBench**, which pairs CuTe DSL and TIRx variants of 74 fused subgraphs under fixed semantics, workloads, hardware, and verification rules. During each variant's solo at-

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.

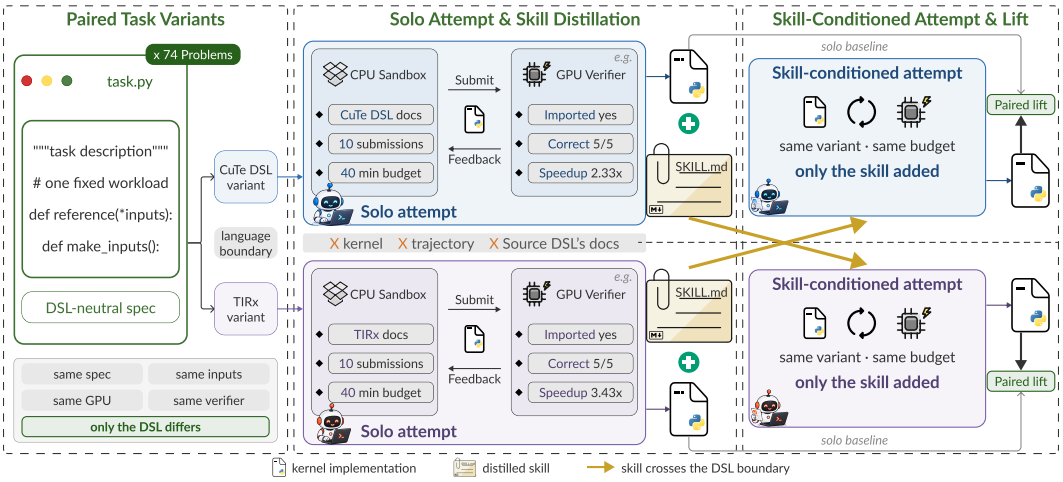


Figure 1: Overview of the MetaKernelBench pipeline. **Left:** one DSL-neutral specification yields paired CuTe DSL and TIRx variants. **Middle:** solo attempts use verifier feedback to optimize implementations and distill findings into natural-language skills. **Right:** only the source skill crosses to a fresh attempt in the other DSL, and paired lift compares its implementation with the solo attempt's.

tempt without transferred knowledge, the agent is instructed to distill optimization findings into a skill. Every available source skill is transferred whether or not the source attempt passes verification, so failed attempts can also contribute useful findings. Each skill is the sole additional input to a fresh skill-conditioned attempt by the same model in the other DSL, with the source implementation and trajectory withheld. We compare each skill-conditioned attempt with the solo attempt on the same variant under matched agent configurations and per-attempt budgets. Paired comparisons assess the distillation and use of skills jointly by measuring the correctness and end-to-end runtime of implementations that may combine eager PyTorch and DSL kernels.

In single-attempt comparisons across six models, paired lift ranges from -19% to $+29\%$ by model and direction. Four models gain in both directions, yet every model and direction has regressions on 16% to 45% of problems. Outcomes follow the source attempt's pass and speed tier relative to the target's solo attempt rather than source success alone. When the source stands above the target's solo attempt, 71% of comparisons improve, and when it stands below, 54% regress. Source skills with unsupported conclusions that rule out alternatives are more common in harmed than in helped comparisons for five of six models, while other content associated with improvements varies across models. A failure analysis finds that larger models are mostly led astray by skills they follow, whereas the two models with the lowest pooled solo pass rates mostly fail in their own execution.

Our contributions are as follows.

- We formulate same-problem cross-DSL kernel knowledge transfer as an evaluation dimension complementary to the correctness and performance of an individual implementation.
- We introduce MetaKernelBench, which pairs 74 problems across CuTe DSL and TIRx, transfers only the source skill regardless of source success, and compares skill-conditioned attempts with solo attempts under matched per-attempt budgets.
- We characterize transfer across six models and both directions, show that outcomes follow the source attempt's standing relative to the target's solo attempt, and relate them to recorded guidance through content, case, and failure analyses.

2 METAKERNELBENCH

MetaKernelBench pairs CuTe DSL and TIRx variants under a shared specification and grading contract, so paired variants differ only in the DSL. It compares each skill-conditioned attempt with the solo attempt on the same variant under matched budgets, so the two differ only in the skill (Figure 1).

2.1 TASK FORMULATION

Each problem p has a DSL-neutral executable specification containing a task description, a PyTorch reference function, and a generator of fresh inputs for a fixed workload configuration. A shared grading contract checks correctness against the reference and measures end-to-end speedup over the reference's eager PyTorch execution. For each kernel DSL $d \in \mathcal{D} = \{\mathrm{CuTe}, \mathrm{TIRx}\}$, the task variant $\tau_{p,d}$ adds only d and its software environment to this specification. The objective is to find the fastest valid implementation using d on a single NVIDIA B200 GPU.

A valid implementation may combine eager PyTorch, ordinary Python, and kernels written in d. Every invocation must execute a DSL kernel that performs a required computation or data-movement step on which at least one returned output tensor depends. A DSL step fails the *bypass test* if bypassing it and returning the existing eager result still yields valid outputs. Other kernel languages and compilers, CUDA Graphs, calls to the reference, and kernels imported from DSL kernel libraries are forbidden. Both DSLs are embedded in Python, so hybrid implementations are their usual form, and the boundary between eager and DSL work is itself an optimization decision. End-to-end timing covers each complete invocation, including Python dispatch, eager operations, and DSL kernels.

CuTe DSL and TIRx both give kernel authors explicit control over pipeline structure, memory placement, synchronization, and hardware roles. CuTe DSL uses layout algebra to partition data and computation across threads (NVIDIA, 2026), whereas TIRx uses storage layouts to guide compiler dispatch of tile primitives (Apache TVM Community, 2026). Kernel code written in one DSL must be re-expressed in the other, while eager PyTorch code runs unchanged in both variants.

2.2 TRANSFER PROTOCOL

A shared directive instructs the agent to maintain a natural-language skill throughout each attempt. The skill should record evidence-backed optimization findings, work partitioning between eager PyTorch and DSL kernels, reasons for failures, applicability conditions, and open questions. The agent is told that this skill may guide a fresh attempt on the same problem in another DSL. Conclusions should not depend on source-DSL syntax or APIs, although details needed to explain evidence are allowed. The directive forbids code, commands, and API instructions.

For source and target DSLs $d_s \neq d_t$ in $\mathcal{D}$, the attempt outputs for direction $d_s \to d_t$ are:

$$(k^{\mathrm{solo}}_{p,d},\, \sigma_{p,d}) \leftarrow \mathcal{A}(\tau_{p,d},\, \emptyset), \quad d \in \mathcal{D}, \qquad (k^{\mathrm{skill}}_{p,\, d_s \to d_t},\, \cdot\,) \leftarrow \mathcal{A}(\tau_{p,d_t},\, \sigma_{p,d_s}), \quad \sigma_{p,d_s} \neq \emptyset. \tag{1}$$

In Equation 1, $\leftarrow$ denotes a fresh execution of agent procedure $\mathcal{A}$ with a fixed model, agent configuration, and per-attempt budget. k and σ denote the final implementation and skill, respectively, $\emptyset$ denotes absence, and the dot stands for the target's own skill, which is not transferred. Each solo attempt supplies the source skill in one direction and is the solo comparator in the other.

The target variant's solo and skill-conditioned attempts are the *baseline* and *treatment*, respectively. The source skill is appended verbatim to the baseline instruction under a neutral header attributing it to a prior attempt on the same problem in another DSL. Every available source skill is transferred whether or not the source implementation passes verification. Transfer does not filter skill content, so the requested format is not enforced. If the source skill is missing or blank, the treatment is not run but is recorded as skipped and inherits the baseline outcome for evaluation.

The baseline and treatment share the same task variant, verifier, feedback mechanism, and implementation rules. The skill attachment is the only additional task input, with the source implementation and trajectory withheld. Matching target budgets does not equalize total computation, since producing the transferred skill also requires a separate source attempt with the same 40-minute budget. Eager PyTorch advice may apply directly, work partitioning may require revision, and kernel strategies require new target-DSL implementations. Paired lift measures the combined end-to-end outcome without isolating the contribution of each type of guidance.

2.3 PROBLEM CONSTRUCTION

MetaKernelBench uses fused subgraphs traceable to production models, serving stacks, or published reference implementations. These subgraphs expose choices about fusion, intermediate materialization, and the division of work between eager PyTorch and DSL kernels. DSL optimization may involve shared-memory and register use as well as overlapping data movement with computation.

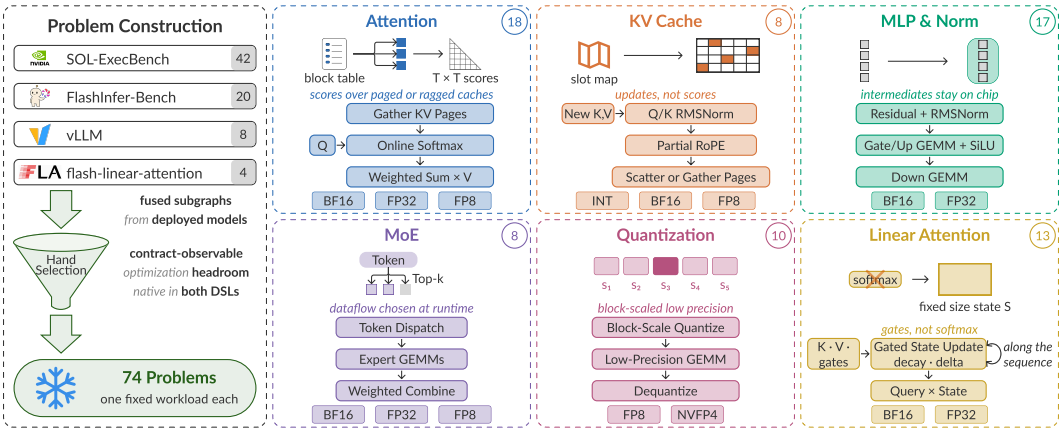


Figure 2: Construction and composition of the MetaKernelBench problem set. **Left:** manual screening of fused subgraphs from four upstream sources yields 74 fixed-workload problems. **Right:** six families show problem counts, representative computation patterns, and numerical formats.

A candidate is retained only if its defining behavior remains observable under the grading contract, it offers meaningful kernel-optimization headroom, and native implementation is feasible in both DSLs. Although subgraphs may span multiple operator types, each is assigned to one of six families according to its main implementation constraint (Figure 2). Attention involves attention scoring, often with paged, ragged, or sparse layouts, whereas KV Cache covers cache maintenance without scoring. MLP & Norm covers normalization and dense fusion chains, while MoE covers routing, dispatch, and expert computation. Quantization centers on block-scaled low-precision computation, while Linear Attention covers recurrent, chunk-scanned, or convolutional sequence mixing.

The 74-problem catalog includes 42 problems adapted from SOL-ExecBench (Lin et al., 2026) and 20 from FlashInfer-Trace and FlashInfer-Bench (Xing et al., 2026). The catalog contains 8 cache-maintenance problems constructed from vLLM tests and kernels (Kwon et al., 2023) and 4 linear-attention problems authored from official model configurations and tests in flash-linear-attention (Yang & Zhang, 2024). Each problem fixes a workload configuration and input distribution, selecting the largest admissible workload when multiple upstream records exist. Where needed, we adapt upstream definitions to expose core behavior, stabilize discrete selections, support reference evaluation with promoted floating-point inputs, and complete standalone call contracts.

3 Experimental Setup

3.1 Models and Agent Scaffold

We evaluate six models, namely DeepSeek-V4-Pro (DeepSeek-AI, 2026), GLM-5.3 (Z.ai, 2026a), GLM-5.3-Flash (Z.ai, 2026b), Gemini 3.7 Flash (Google, 2026), GPT-5.6 Sol (OpenAI, 2026), and Qwen3.8-Flash (Qwen Team, 2026). The panel contains three Flash-tier models and three larger models, with GLM-5.3 and GLM-5.3-Flash forming a same-family pair. All models use Terminal-Bench's terminal-only Terminus-2 scaffold (Merrill et al., 2026) through Harbor 0.20.0 (Harbor Framework Team, 2026). Context compression reintroduces the complete task instruction, including any attached source skill. We do not override sampling temperature or reasoning effort in either arm.

3.2 Execution Environment

Each attempt runs in an isolated CPU-only sandbox with two cores, 8 GB of memory, and no network access. The sandbox provides the task specification, grading contract, submission script, fixed-version dependencies for the variant's DSL, and offline documentation. The agent has a 40-minute wall-clock budget, including feedback waits, and at most 10 feedback submissions. Each submission snapshots the implementation for evaluation in a fresh container on an NVIDIA B200, using new input seeds and the sandbox's DSL dependencies. Feedback contains import and correctness

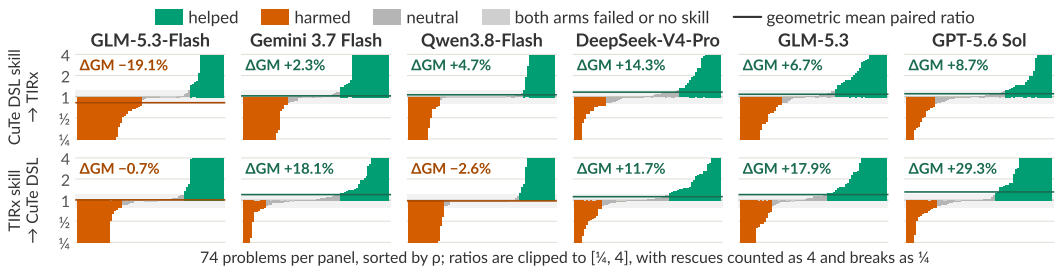


Figure 3: Problem-level paired ratios by model and transfer direction.

results, errors, and, on success, the end-to-end implementation and eager-reference times with their ratio. At budget expiry or an early stop, any collected final implementation is evaluated afresh on a B200. Each phase permits up to three tries for transient infrastructure failures, with execution retries restarting the agent and final-evaluation retries re-evaluating only the collected implementation.

3.3 Evaluation Metrics

A scored pass requires float64-anchored correctness and completed end-to-end timing. The verifier compares floating-point outputs against a reference evaluated with promoted inputs, using the original reference's error to set tolerances. Each evaluation uses five correctness draws that must all pass, 10 warmup calls, and 50 timed calls, with correctness rechecked on three timed outputs. A static check rejects implementations using TIRx's inline CUDA source entry point, which embeds raw CUDA. The size of the DSL contribution is not scored, since runtime already reflects its value.

Solo metrics use final outcomes by model and DSL, including attempts that produced no skill. Speedup is the eager-reference median runtime divided by the implementation's median runtime, capped at $100\times$. We report pass rate P, passing-only geometric mean speedup (GM), and the $2\times$-or-better pass rate $R(2)$, retaining failures in both rate denominators.

A comparison i is resolved when its source and both target arms have final outcomes, with a skipped treatment inheriting the baseline outcome. For arm $a \in \{\text{base}, \text{treat}\}$, let $p_i^a \in \{0, 1\}$ indicate passing and t_i^a be its end-to-end median runtime when passing, giving the paired ratio:

$$\rho_i = \text{clip}\big(t_i^{\text{base}}/t_i^{\text{treat}},\ \tfrac{1}{4},\ 4\big) \text{ if } p_i^{\text{base}} p_i^{\text{treat}} = 1, \qquad \rho_i = 4^{\,p_i^{\text{treat}} - p_i^{\text{base}}} \text{ otherwise.} \tag{2}$$

Equation 2 uses raw runtimes, not capped speedups, for jointly passing arms and assigns 4 to failure-to-pass rescues, $1/4$ to pass-to-failure breaks, and 1 to joint failures or skipped treatments. The transition values are scores rather than measured speedups, placed at the clip bound so that a verdict change weighs as much as the largest retained runtime change.

Over a set G of resolved comparisons, we aggregate ρ_i as paired lift, help rate, and harm rate:

$$\Delta\text{GM}_G = \exp\Big(\frac{1}{|G|}\sum_{i\in G} \ln \rho_i\Big) - 1, \quad T_G = \frac{1}{|G|}\sum_{i\in G} \mathbf{1}[\rho_i \geq \tfrac{5}{4}], \quad H_G = \frac{1}{|G|}\sum_{i\in G} \mathbf{1}[\rho_i \leq \tfrac{4}{5}]. \tag{3}$$

Equation 3 weights comparisons equally, leaving a neutral share of $1 - T_G - H_G$. We report ΔGM_G, T_G, and H_G as percentages by model and direction, within each family and over all 74 problems.

4 Results

4.1 Overall Transfer Outcomes

Skill conditioning yields paired lifts from -19.1% to $+29.3\%$ across six models and both directions in single-attempt outcomes (Table 1). Gemini 3.7 Flash, DeepSeek-V4-Pro, GLM-5.3, and GPT-5.6 Sol gain in both directions, by 2.3% to 29.3%. GLM-5.3-Flash has negative lift in both directions, while Qwen3.8-Flash gains 4.7% from CuTe DSL skills and loses 2.6% from TIRx skills, with its 43 missing-skill treatments skipped and scored as neutral. Help and harm occur for every

Table 1: Cross-DSL skill transfer against the target variant's solo attempts. Cells report help rate T, paired lift ΔGM, and harm rate H in percent, with green bars showing T, orange bars showing H, and the gap the neutral share. Underlining marks the highest ΔGM in each direction and column.

Model T/ΔGM/H	**Attention** (18)	**KV Cache** (8)	**Linear Attn** (13)	**MLP & Norm** (17)	**MoE** (8)	**Quant** (10)	**All** (74)
CuTe DSL skill ⟶ fresh TIRx attempt							
GLM-5.3-Flash	22.2/−17.3/50.0	25.0/−37.8/62.5	23.1/−28.8/53.8	17.6/−16.2/35.3	37.5/+14.0/37.5	20.0/−19.1/30.0	23.0/−19.1/44.6
Gemini 3.7 Flash	44.4/+26.0/27.8	12.5/−0.9/25.0	46.2/−3.2/38.5	23.5/+0.3/23.5	25.0/+2.2/37.5	40.0/−19.9/40.0	33.8/+2.3/31.1
Qwen3.8-Flash	22.2/+6.3/27.8	25.0/−21.4/37.5	15.4/+6.1/15.4	11.8/−0.3/17.6	37.5/+68.2/0.0	30.0/−6.5/30.0	21.6/+4.7/21.6
DeepSeek-V4-Pro	44.4/+22.2/11.1	50.0/+84.4/12.5	23.1/−5.7/46.2	5.9/−9.3/29.4	12.5/+3.0/12.5	40.0/+43.0/10.0	28.4/+14.3/21.6
GLM-5.3	44.4/+8.2/33.3	12.5/−40.9/62.5	61.5/+73.3/15.4	17.6/−13.6/29.4	12.5/+4.1/12.5	50.0/+29.4/20.0	35.1/+6.7/28.4
GPT-5.6 Sol	38.9/+4.2/33.3	50.0/+32.2/12.5	23.1/−7.4/30.8	11.8/−9.8/11.8	12.5/−10.8/25.0	70.0/+99.1/0.0	32.4/+8.7/20.3
TIRx skill ⟶ fresh CuTe DSL attempt							
GLM-5.3-Flash	38.9/+20.2/22.2	0.0/−44.6/62.5	23.1/+1.0/23.1	11.8/−0.9/29.4	25.0/−28.4/50.0	60.0/+43.4/20.0	27.0/−0.7/31.1
Gemini 3.7 Flash	38.9/+7.5/33.3	62.5/+59.3/12.5	53.8/+56.0/0.0	5.9/−1.8/17.6	37.5/+20.2/0.0	20.0/+3.7/20.0	33.8/+18.1/16.2
Qwen3.8-Flash	16.7/−7.4/22.2	25.0/−30.3/50.0	15.4/−19.3/30.8	47.1/+44.6/11.8	12.5/−17.7/25.0	20.0/+4.0/20.0	24.3/−2.6/24.3
DeepSeek-V4-Pro	33.3/+5.2/16.7	50.0/+11.8/37.5	38.5/+32.7/23.1	41.2/+12.0/5.9	25.0/−5.7/25.0	20.0/+13.4/20.0	35.1/+11.7/18.9
GLM-5.3	55.6/+64.8/11.1	62.5/+67.9/25.0	23.1/−19.4/38.5	41.2/+18.3/11.8	37.5/+27.9/0.0	20.0/−25.6/50.0	40.5/+17.9/21.6
GPT-5.6 Sol	50.0/+40.4/33.3	25.0/+4.9/37.5	38.5/+40.0/15.4	29.4/+20.6/5.9	25.0/+20.6/25.0	60.0/+41.1/20.0	39.2/+29.3/21.6

Table 2: Solo implementation quality. Cells report pass rate P, passing-only geometric mean speedup GM over the eager reference, and $2\times$-or-better pass rate $R(2)$. Speedups are capped at $100\times$ per attempt. Underlining marks the highest GM in each DSL and column.

Model P/GM×/$R(2)$	**Attention** (18)	**KV Cache** (8)	**Linear Attn** (13)	**MLP & Norm** (17)	**MoE** (8)	**Quant** (10)	**All** (74)
CuTe DSL							
GLM-5.3-Flash	50.0/5.7×/38.9	100.0/1.5×/50.0	69.2/6.6×/38.5	100.0/1.7×/35.3	62.5/1.5×/12.5	60.0/3.6×/40.0	73.0/2.7×/36.5
Gemini 3.7 Flash	94.4/9.9×/66.7	100.0/12.0×/100.0	92.3/17.7×/76.9	100.0/2.6×/47.1	100.0/1.6×/25.0	100.0/3.5×/40.0	97.3/5.7×/59.5
Qwen3.8-Flash	50.0/8.2×/33.3	75.0/5.1×/50.0	61.5/11.8×/46.2	70.6/1.8×/17.6	62.5/1.5×/12.5	80.0/5.1×/60.0	64.9/4.4×/35.1
DeepSeek-V4-Pro	100.0/1.9×/27.8	100.0/3.1×/50.0	100.0/2.4×/30.8	100.0/1.3×/11.8	100.0/1.4×/25.0	100.0/1.4×/20.0	100.0/1.8×/25.7
GLM-5.3	77.8/5.0×/44.4	100.0/5.0×/62.5	100.0/13.6×/69.2	100.0/2.0×/41.2	100.0/2.0×/37.5	70.0/6.0×/60.0	90.5/4.4×/51.4
GPT-5.6 Sol	100.0/2.6×/38.9	100.0/14.0×/100.0	100.0/8.7×/69.2	100.0/1.9×/35.3	100.0/1.9×/50.0	90.0/7.3×/80.0	98.6/3.9×/56.8
TIRx							
GLM-5.3-Flash	66.7/13.1×/55.6	100.0/7.7×/87.5	69.2/7.8×/46.2	100.0/1.7×/41.2	37.5/1.5×/12.5	80.0/4.8×/70.0	77.0/4.7×/51.4
Gemini 3.7 Flash	55.6/16.2×/38.9	62.5/15.9×/62.5	76.9/15.7×/61.5	52.9/2.2×/23.5	62.5/2.3×/25.0	80.0/8.3×/60.0	63.5/7.9×/43.2
Qwen3.8-Flash	55.6/7.2×/44.4	87.5/17.3×/87.5	61.5/34.0×/53.8	70.6/3.4×/41.2	12.5/3.9×/12.5	80.0/6.9×/80.0	62.2/8.7×/51.4
DeepSeek-V4-Pro	100.0/2.1×/33.3	100.0/2.8×/62.5	100.0/6.3×/53.8	100.0/1.9×/41.2	100.0/1.6×/25.0	100.0/1.6×/30.0	100.0/2.4×/40.5
GLM-5.3	83.3/10.7×/61.1	100.0/16.5×/100.0	100.0/8.9×/61.5	100.0/2.7×/52.9	100.0/2.4×/50.0	90.0/6.8×/70.0	94.6/6.2×/63.5
GPT-5.6 Sol	100.0/6.6×/61.1	87.5/15.6×/87.5	100.0/17.1×/84.6	100.0/2.5×/47.1	100.0/2.8×/62.5	90.0/6.4×/60.0	97.3/6.1×/64.9

model and direction (Figure 3), at rates of 21.6% to 40.5% and 16.2% to 44.6%, respectively. Verdict changes account for 76 of the 277 helped and 75 of the 223 harmed comparisons, so most help and harm lie in runtime differences between jointly passing attempts.

Paired lifts over all 74 problems pool families with opposite signs in 11 of the 12 model and direction cells. For GLM-5.3, TIRx skills yield 67.9% lift on KV Cache while CuTe DSL skills yield -40.9%, with the signs reversed on Linear Attention. GPT-5.6 Sol's positive CuTe DSL to TIRx lift of 8.7% rests on the 10 Quantization problems, where its lift is 99.1% against -1.1% on the other 64. Excluding each cell's best family turns this lift and three of the other four positive lifts from CuTe DSL skills negative, whereas all four positive lifts from TIRx skills stay positive.

Among the 277 helped comparisons, the reverse direction on the same model and problem is harmed in 41.5% and helped in 23.1%, with harm exceeding help for each of the six models. Without transfer, the reverse direction's skill-conditioned attempt is another solo attempt, so that comparison would be helped and harmed equally often regardless of the forward outcome. The imbalance therefore reflects the transferred skills, not chance, and a skill's value depends on its source attempt.

4.2 Implementation Quality and Transfer Outcomes

Solo pass rates and passing-only speed rank the six models differently (Table 2). DeepSeek-V4-Pro passes every variant but has the lowest GMs, $1.8\times$ for CuTe DSL and $2.4\times$ for TIRx. Qwen3.8-Flash

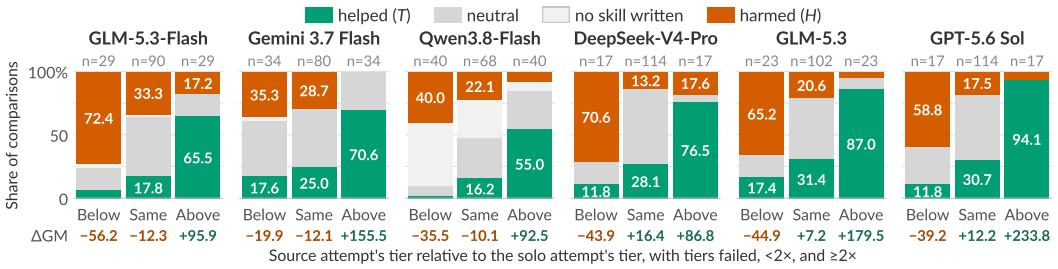


Figure 4: Transfer outcomes by the source attempt's tier relative to the solo attempt's. Each panel pools a model's comparisons in both directions, with skips counted as neutral and ΔGM in percent. Only helped and harmed segments of at least 10% carry value labels.

More in harmed / More in helped	Agenda handoff 41% of skills	Unsupported closure 16% of skills	Unmeasured prescriptions 9% of skills	Source-DSL mechanics 70% of skills	Verification recipes 29% of skills	Bounded failure diagnosis 78% of skills	Workload census 84% of skills	Ambition capping 53% of skills	Bottleneck and timing evidence 86% of skills	Verified reformulation 54% of skills	Semantic and precision hazards 71% of skills
GLM-5.3-Flash 37 helped, 56 harmed	0	−13	−4	+2	−4	0	−13	+21	+3	0	+8
Gemini 3.7 Flash 50 helped, 35 harmed	0	−7	+1	−8	−2	−13	+12	−3	0	+9	−3
Qwen3.8-Flash 34 helped, 34 harmed	−1	−17	−19	−7	+3	+8	0	−5	0	−2	+6
DeepSeek-V4-Pro 47 helped, 30 harmed	−11	−8	+2	−2	−9	−4	+12	−11	+4	+7	+3
GLM-5.3 56 helped, 37 harmed	−27	0	+5	−4	−1	+3	−2	+6	+2	+9	+1
GPT-5.6 Sol 53 helped, 31 harmed	−15	−7	0	+4	−1	−3	−6	−3	0	−12	+3
Mean of models 6 models	**−9**	**−9**	**−3**	**−2**	**−2**	**−2**	**+1**	**+1**	**+1**	**+2**	**+3**

Figure 5: Skill content across transfer outcomes. Cells show a label's prevalence in helped comparisons minus that in harmed comparisons, in percentage points averaged over the two directions. Percentages under category names give each label's prevalence over all 841 labeled skills.

has the lowest pass rates, 64.9% and 62.2%, yet the highest TIRx GM of 8.7×. For every model, TIRx implementations have a higher GM than CuTe DSL ones, while pass rates favor neither DSL. Passing-only GMs summarize different solved subsets across models and depend on the efficiency of each eager reference, whose runtime includes Python-level iteration in recurrent workloads. Gemini 3.7 Flash's scored TIRx pass rate excludes 21 verifier-passing implementations using inline CUDA. The four models with the highest solo pass rates pooled across both DSLs are the four with positive paired lift in both directions (Table 1), whereas solo speed orders the models differently from their lifts. Six models are too few to conclude that solo pass rate predicts paired lift.

Figure 4 groups comparisons by whether the source attempt's outcome tier lies below, at, or above that of the solo attempt, with tiers failed, below 2×, and at 2× or better. Pooled over models and directions, 71.2% of comparisons are helped and 8.1% harmed with the source above, 25.7% and 21.8% at the same tier, and 10.6% and 53.8% with it below. Help rate follows this ordering in every model, with 55.0% to 94.1% helped when the source stands above and 35.3% to 72.4% harmed when it stands below. At the same tier, the three larger models retain lifts of 7.2% to 16.4%. Skill-conditioned outcomes thus track the source attempt's standing rather than its verdict, so a passing source does not guarantee beneficial reuse and a failed source does not preclude it.

4.3 Skill Content across Transfer Outcomes

LLM annotators label each source skill against a fixed, hand-built 11-category rubric. Its categories range from workload facts and verified rewrites to unsupported conclusions and open agendas. Unsupported closure, which rules out alternatives beyond the stated evidence, is 7 to 17 points more prevalent in harmed than in helped comparisons for five of six models (Figure 5). Agenda handoff, which leaves optimization suggestions open for later work, is 11 to 27 points more prevalent in harmed than in helped comparisons for the three larger models.

Verified reformulation, a rewrite reported as implemented and correct, is 7 to 9 points more prevalent in helped than in harmed comparisons for Gemini 3.7 Flash, DeepSeek-V4-Pro, and GLM-5.3, but 12 points more in harmed comparisons for GPT-5.6 Sol. Ambition capping, which advises minimal-

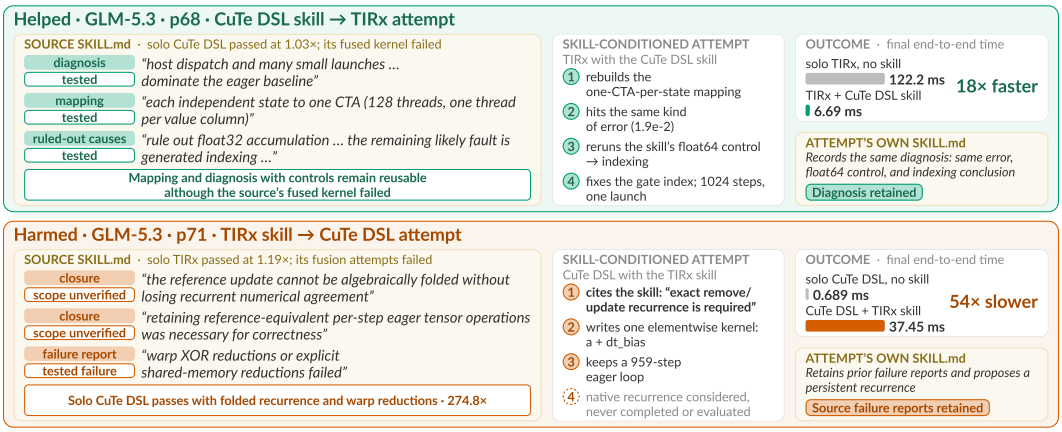


Figure 6: Two GLM-5.3 transfers from passing source implementations. **Top:** CuTe DSL to TIRx on p68, helped, reusing a thread mapping and a diagnostic control. **Bottom:** TIRx to CuTe DSL on p71, harmed, retaining an eager recurrence. Tags mark whether a graded source submission supports each claim, and runtimes are unclipped end-to-end times against solo.

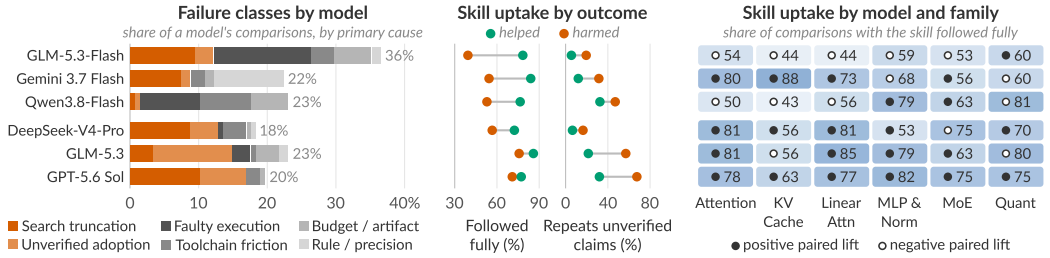


Figure 7: Failure causes and skill uptake. **Left:** primary causes of harm as shares of a model's 148 comparisons. **Middle:** shares of skill-conditioned attempts that follow the source skill fully or repeat its unverified claims in their own skill. **Right:** the share followed fully by family.

ity or delegation without measured justification, is 21 points more prevalent in helped comparisons for GLM-5.3-Flash but 11 points more in harmed comparisons for DeepSeek-V4-Pro.

Figure 6 traces two GLM-5.3 transfers from passing source implementations, one helped and one harmed. The p68 source skill records verification recipes and a bounded failure diagnosis, and the TIRx skill-conditioned attempt reuses the recorded thread mapping, reruns the recorded control, fixes the indexing fault the source had suspected, fuses the recurrence the source could not, and records the same diagnosis in its own skill. The p71 source skill records an unsupported closure that rules out folding the recurrence, and the CuTe DSL skill-conditioned attempt cites it, retains eager execution, and never completes the native recurrence that the solo attempt implements. These traces show skill uptake, but the runtime comparisons do not isolate its causal contribution.

4.4 Failure Causes and Skill Uptake

LLM annotators audit each skill-conditioned attempt's trajectory for its uptake of the source skill and for unverified source claims repeated in its own skill, and assign one primary cause from a six-class taxonomy to each harmed comparison with a target-side failure (Figure 7). Search truncation, in which the attempt reproduces the source design and never explores a better one its DSL allows, is the largest cause for DeepSeek-V4-Pro and GPT-5.6 Sol, at 8.8% and 10.1% of their comparisons. Unverified adoption of a wrong or out-of-scope source claim, most often a prohibition the attempt never tests, is the largest for GLM-5.3, at 11.5%. Faulty execution of a sound plan is the largest cause for GLM-5.3-Flash and Qwen3.8-Flash, at 14.2% and 8.8%. Rule and precision failure, mostly exclusions for inline CUDA, is the largest for Gemini 3.7 Flash, at 10.1%. The two

skill-side classes, truncation and unverified adoption, account for 65% to 86% of the assigned causes for the three larger models and 39% for Gemini 3.7 Flash, against 33% for GLM-5.3-Flash and 6% for Qwen3.8-Flash. For these two models, faulty execution, toolchain friction, and budget and artifact mismanagement account for 63% and 94%, so the larger models are led astray by the skill while the two models with the lowest pooled solo pass rates fail in their own execution.

Harm arises while the skill is in use, since the skill is followed fully or partially in over 80% of harmed comparisons, and fully in 39% to 76% of them, against 72% to 86% of helped comparisons (Figure 7, middle). The three larger models follow the skill fully 6 to 16 points less often in harmed than in helped comparisons, whereas the three Flash-tier models follow it 24 to 39 points less often, so the models that the skill leads astray are those that follow it most faithfully when it does harm. The share followed fully also varies by family, from 77% to 85% on Attention and Linear Attention against 56% to 75% on KV Cache and MoE for the three larger models, stays between 44% and 60% for GLM-5.3-Flash, and has a rank correlation of 0.49 with paired lift over the 36 model and family cells (Figure 7, right). For every model, unverified source claims reappear in the attempt's own skill more often in harmed than in helped comparisons, in 17% to 68% against 5% to 32% of cases, so unverified claims propagate into the next generation of skills. Uptake thus tracks benefit, yet it is also the channel through which such claims propagate.

5 Related Work

Kernel generation and benchmarks. LLMs optimize GPU kernels through model adaptation and execution-guided search (Yu et al., 2026). Reinforcement learning and domain-specific fine-tuning train specialized kernel generators (Li et al., 2026; Baronio et al., 2026; Dai et al., 2026; Bai et al., 2026). Agents refine candidate kernels through compilation, execution, and profiling (Wei et al., 2025; Zhang et al., 2025), while evolutionary methods search populations with execution feedback (Wiedemann et al., 2026; Chen et al., 2026). KernelBench evaluates LLM-generated kernels by correctness and speedup over PyTorch (Ouyang et al., 2025). Subsequent benchmarks target Triton (Li et al., 2025), harden verification against reward hacking (Lange et al., 2025; Zhang et al., 2026b), and combine software baselines with hardware-derived performance limits (Lin et al., 2026). Agentic harnesses compare workflows across models and languages (Guan et al., 2026; Xing et al., 2026), while other suites cover additional accelerator platforms and mobile devices (Wen et al., 2026; Nie et al., 2026; Zou et al., 2026). AgentKernelArena tests kernels on unseen input configurations to evaluate implementation generalization (Younesian et al., 2026).

Code and experience reuse. Experience is retained in persistent knowledge bases (Dong et al., 2026; Liao et al., 2026), trajectory-derived reasoning graphs (Gong et al., 2026), and literature-derived skills (Sun et al., 2026). At the code level, CASS-Bench evaluates source- and assembly-level translation of kernels across vendor stacks (Heakl et al., 2026), and CuBridge adapts expert CUDA kernels through an executable intermediate representation (Ma et al., 2026). KLineage extracts verified, code-anchored skills from expert kernel lineages for reuse in CUDA and TileLang (Zhang et al., 2026a). daVinci-kernel admits self-distilled skills only after reinjection meets an execution-based speedup threshold on the original task (Fu et al., 2026).

MetaKernelBench transfers only unscreened self-distilled skills from passing and failed source attempts and tests whether they help or harm fresh attempts on the same problem in another DSL.

6 Conclusion

MetaKernelBench evaluates same-problem cross-DSL kernel knowledge transfer by comparing solo attempts with skill-conditioned attempts whose only extra input is a source skill, under matched per-attempt budgets. Four of six models gain in both directions, yet every model and direction shows both help and harm. Outcomes follow the source attempt's standing relative to the target's solo attempt, not source success alone, improving more often when the source stands above and regressing more often when it stands below. For five of six models, unsupported conclusions that rule out alternatives are more common in harmed than in helped comparisons, while content tied to improvements varies by model. Source implementation quality alone does not determine reuse value, which depends on the source's standing against the target, so both must be measured.

AI USE STATEMENT

The research question, the benchmark, and all design and analysis decisions are the authors' own. Under the authors' direction, generative AI tools assisted in implementing the methods, cleaning and reformatting the problem set, and polishing the text. In addition, LLMs served as the annotators described in the paper. The authors tested the code, verified the problem set, recomputed all reported numbers, and take full responsibility for the final content.

REFERENCES

Apache TVM Community. Tirx: An open compiler stack for evolving frontier ml kernels. https://tvm.apache.org/2026/06/22/tirx, 2026.

Haolei Bai, Lingcheng Kong, Xueyi Chen, Jiamian Wang, Zhiqiang Tao, and Huan Wang. Dice: Diffusion large language models excel at generating cuda kernels. *arXiv preprint arXiv:2602.11715*, 2026.

Carlo Baronio, Pietro Marsella, Ben Pan, Simon Guo, and Silas Alberti. Kevin: Multi-turn rl for generating cuda kernels. In *ICLR*, 2026.

Terry Chen, Zhifan Ye, Bing Xu, Zihao Ye, Timmy Liu, Ali Hassani, Tianqi Chen, Andrew Kerr, Haicheng Wu, Yang Xu, Yu-Jung Chen, Hanfeng Chen, Aditya Kane, Ronny Krashinsky, Ming-Yu Liu, Vinod Grover, Luis Ceze, Roger Bringmann, John Tran, Wei Liu, Fung Xie, Michael Lightstone, and Humphrey Shi. Avo: Agentic variation operators for autonomous evolutionary search. *arXiv preprint arXiv:2603.24517*, 2026.

Weinan Dai, Hanlin Wu, Qiying Yu, Huan-ang 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. Cuda agent: Large-scale agentic rl for high-performance cuda kernel generation. *arXiv preprint arXiv:2602.24286*, 2026.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Kris Shengjun Dong, Sahil Modi, Dima Nikiforov, Sana Damani, Edward Lin, Siva Kumar Sastry Hari, and Christos Kozyrakis. Kernelblaster: Continual cross-task cuda optimization via memory-augmented in-context reinforcement learning. *arXiv preprint arXiv:2602.14293*, 2026.

Dayuan Fu, Mohan Jiang, Tongyu Wang, Dian Yang, Jiarui Hu, Liming Liu, Jinlong Hou, and Pengfei Liu. davinci-kernel: Co-evolving skill selection, summarization, and utilization via rl for gpu kernel optimization. *arXiv preprint arXiv:2606.16497*, 2026.

Junfeng Gong, Zhiyi Wei, Junying Chen, Cheng Liu, and Huawei Li. From large to small: Transferring cuda optimization expertise via reasoning graph. In *ICLR*, 2026.

Google. Introducing gemini 3.7 flash. https://blog.google/innovation-and-ai/models-and-research/gemini-models/introducing-gemini-3-7-flash/, 2026.

Yue Guan, Yichen Lin, Xu Zhao, Jianzhu Yao, Xinwei Qiang, Zhongkai Yu, Pramod Viswanath, Yufei Ding, and Adnan Aziz. Tritongym: A benchmark for agentic llm workflows in triton gpu code generation. In *ICML*, 2026.

Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments. https://github.com/harbor-framework/harbor, 2026.

Ahmed Heakl, Gustavo Bertolo Stahl, Sarim Hashmi, Seung Hun Eddie Han, Mukul Ranjan, Arina Kharlamova, Salman Khan, and Abdulrahman Mahmoud. Cass: Nvidia to amd transpilation with data, models, and benchmark. In *ACL*, 2026.

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 *SOSP*, 2023.

Robert Tjarko Lange, Qi Sun, Aaditya Prasad, Maxence Faldor, Yujin Tang, and David Ha. Towards robust agentic cuda kernel benchmarking, verification, and optimization. *arXiv preprint arXiv:2509.14279*, 2025.

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 *ACL Findings*, 2025.

Xiaoya Li, Xiaofei Sun, Albert Wang, Jiwei Li, and Chris Shum. Cuda-l1: Improving cuda optimization via contrastive reinforcement learning. In *ICLR*, 2026.

Gang Liao, Hongsen Qin, Ying Wang, Alicia Golden, Michael Kuchnik, Yavuz Yetim, Ruichao Xiao, Jia Jiunn Ang, Chunli Fu, Yihan He, Samuel Hsia, Zewei Jiang, Roman Levenstein, Dianshi Li, Liyuan Li, Ajit Mathews, Varna Puvvada, Feng Shi, Nathan Yan, Xiayu Yu, Uladzimir Pashkevich, Matt Steiner, Carole-Jean Wu, and Gaoxiang Liu. Kernelevolve: Scaling agentic kernel coding for heterogeneous ai accelerators at meta. In *ISCA*, 2026.

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 Bringmann, Cyril Zeller, Michael Lightstone, Christos Kozyrakis, and Humphrey Shi. Sol-execbench: Speed-of-light benchmarking for real-world gpu kernels against hardware limits. *arXiv preprint arXiv:2603.19173*, 2026.

Xing Ma, Yangjie Zhou, Wu Sun, Zihan Liu, Jingwen Leng, Yun Lin, Shixuan Sun, Minyi Guo, and Jin Song Dong. Cubridge: An llm-based framework for understanding and reconstructing high-performance attention kernels. In *ACL*, 2026.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Jenia Jitsev, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *ICLR*, 2026.

Jiayi Nie, Haoran Wu, Yao Lai, Zeyu Cao, Cheng Zhang, Binglei Lou, Erwei Wang, Jianyi Cheng, Timothy M. Jones, Robert Mullins, Rika Antonova, and Yiren Zhao. Kernelcraft: Benchmarking for agentic close-to-metal kernel generation on emerging hardware. In *ICML*, 2026.

NVIDIA. Overview — nvidia cutlass documentation. `https://docs.nvidia.com/cutlass/4.7.0/media/docs/pythonDSL/overview.html`, 2026.

OpenAI. Gpt-5.6: Frontier intelligence that scales with your ambition. `https://openai.com/index/gpt-5-6/`, 2026.

Anne Ouyang, Simon Guo, Simran Arora, Alex L Zhang, William Hu, Christopher Ré, and Azalia Mirhoseini. Kernelbench: Can llms write efficient gpu kernels? In *ICML*, 2025.

Qwen Team. Qwen3.8-flash-next: A new architecture, towards ultimate cost-efficiency. `https://qwen.ai/blog?id=qwen3.8-flash-next`, 2026.

Qitong Sun, Jun Han, Tianlin Li, Zhe Tang, Sheng Chen, Fei Yang, Aishan Liu, Xianglong Liu, and Yang Liu. Kernelskill: A multi-agent framework for gpu kernel optimization. *arXiv preprint arXiv:2603.10085*, 2026.

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. *arXiv preprint arXiv:2507.23194*, 2025.

Anjiang Wei, Tianran Sun, Yogesh Seenichamy, Hang Song, Anne Ouyang, Azalia Mirhoseini, Ke Wang, and Alex Aiken. Astra: A multi-agent system for gpu kernel performance optimization. *arXiv preprint arXiv:2509.07506*, 2025.

Zhongzhen Wen, Yinghui Zhang, Zhong Li, Zhongxin Liu, Linna Xie, and Tian Zhang. Multikernelbench: A multi-platform benchmark for kernel generation. In *ASE*, 2026.

Nina Wiedemann, Quentin Leboutet, Michael Paulitsch, Diana Wofk, and Benjamin Ummenhofer. Kernelfoundry: Hardware-aware evolutionary gpu kernel optimization. In *ICML*, 2026.

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. In *MLSys*, 2026.

Songlin Yang and Yu Zhang. Fla: A triton-based library for hardware-efficient implementations of linear attention mechanism. `https://github.com/fla-org/flash-linear-attention`, 2024.

Sharareh Younesian, Wenwen Ouyang, Sina Rafati, Mehdi Rezagholizadeh, Sharon 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. *arXiv preprint arXiv:2605.16819*, 2026.

Yang Yu, Peiyu Zang, Chi Hsu Tsai, Haiming Wu, Yixin Shen, Jialing Zhang, Haoyu Wang, Zhiyou Xiao, Jingze Shi, Yuyu Luo, Wentao Zhang, Chunlei Men, Guang Liu, and Yonghua Lin. Towards automated kernel generation in the era of llms. In *IJCAI*, 2026.

Z.ai. Glm-5.3: Frontier coding with emergent cyber capabilities. `https://z.ai/blog/glm-5.3`, 2026a.

Z.ai. Glm-5.3-flash: Frontier intelligence, flash cost. `https://z.ai/blog/glm-5.3-flash`, 2026b.

Shuoming Zhang, Qiuchu Yu, Yangyu Zhang, Ruiyuan Xu, Xiyu Shi, Guangli Li, Xiaobing Feng, Huimin Cui, and Jiacheng Zhao. Learning when to optimize: Verified optimization skills from expert gpu-kernel lineages. *arXiv preprint arXiv:2605.28213*, 2026a.

Yunxiang Zhang, Ping Yu, Jianyu Wang, Max (Xiangjun) Fan, Julian Reed, Azalia Mirhoseini, and Will Su. Kernelbench-verified: Do llm-generated kernels actually beat pytorch? *arXiv preprint arXiv:2607.16241*, 2026b.

Zijian Zhang, Rong Wang, Shiyang Li, Yuebo Luo, Mingyi Hong, and Caiwen Ding. Cudaforge: An agent framework with hardware feedback for cuda kernel optimization. *arXiv preprint arXiv:2511.01884*, 2025.

Xingze Zou, Jing Wang, Yuhua Zheng, Xueyi Chen, Haolei Bai, Lingcheng Kong, Syed A.R. Abu-Bakar, Zhaode Wang, Chengfei Lv, Haoji Hu, and Huan Wang. Mobilekernelbench: Can llms write efficient kernels for mobile devices? *arXiv preprint arXiv:2603.11935*, 2026.

A Experimental Details

A.1 Task Instruction

Figure 8 excerpts the task instruction that opens every attempt. Its deliverable, implementation boundary, and grading sections state the rules a submission must follow, while its suggested workflow section only advises. The instruction does not vary by model. A problem's two prompts differ only in the DSL they name, and their sandboxes only in that DSL's documentation and dependencies.

A.2 Workloads and Runtime Configuration

Within each problem's fixed workload configuration, tensor values vary across draws, and some generators also vary index structures and sequence lengths. An implementation may therefore specialize only to properties that hold in every legal draw. Each problem file records its upstream source, reference implementation, workload configuration, and license.

Both variants build on the PyTorch 2.13.0 image with CUDA 13.2 and cuDNN 9. CuTe DSL uses CUTLASS DSL 4.7.0, TIRx uses Apache TVM 0.25.0.post1, and both use Apache TVM FFI 0.1.12. Each single-use B200 grading container has four CPU cores and 16 GB of host memory, with network access disabled. Environment setup and final evaluation lie outside the 40-minute agent budget.

A.3 Correctness and Timing

The anchor z is computed by re-evaluating the reference with float32, float16, and bfloat16 input tensors promoted to float64. Other input dtypes and explicit casts inside the reference remain unchanged. Each floating-point output tensor must satisfy:

$$\|u - z\|_\infty \le 3 \max\{\|v - z\|_\infty, \epsilon \|z\|_\infty\}. \tag{4}$$

In Equation 4, u and v are the submitted and reference outputs, and ϵ is the machine epsilon of their shared dtype. The submitted outputs must match the reference in container structure, shape, dtype, and device, and integer and Boolean outputs must match exactly.

Each evaluation draws fresh correctness, warmup, and timing seeds, and generates separate implementation and reference inputs from matching seeds. Input generation, reference evaluation, and eager-reference timing disable TF32 and reduced-precision reductions, set float32 matmul precision to highest, and use float32 as the default dtype. These settings are not imposed on the submitted implementation. Speedups over the eager reference therefore include this difference in numerics, while paired lift compares two implementations and does not.

Each complete call is timed with CUDA events after device synchronization, excluding input generation and correctness checks. Implementation import, correctness checks, warmup, timing, and rechecks share a 600-second child-process limit that excludes the eager-reference timing.

A.4 Implementation Rule Check

During analysis, a static check scores final implementations as nonpassing when their parsed source calls TIRx's inline CUDA entry point with a `source_code` keyword argument. It rejects 42 verifier-passing implementations, comprising 41 from Gemini 3.7 Flash, of which 21 are solo and 20 skill-conditioned, and one solo implementation from GLM-5.3-Flash.

Without this check, paired lift over all 74 problems retains its sign in every model and direction, and the ranges of help and harm rates in Section 4.1 remain unchanged. The largest lift change is Gemini 3.7 Flash's CuTe DSL to TIRx lift from 2.3% to 0.6%, while the pooled help and harm rates by relative standing in Section 4.2 change by less than four percentage points.

instruction.md · one prompt per task variant, rendered from a single template

⟨DSL⟩ = CuTe DSL or TIRx | GPU = B200 (sm_100a) | budget = 40 minutes | 10 feedback submissions | sandbox = 2 cores, 8 GB, no GPU, no network | kernel.py ≤ 300,000 bytes

Task	*"Implement that computation for an NVIDIA B200, satisfy the correctness contract below, and make it run **as quickly as possible**."* *"every call to solve(…) must execute **at least one ⟨DSL⟩ kernel** that performs a required step of the computation … on which at least one returned output tensor depends. The ⟨DSL⟩ contribution **may be arbitrarily small**; its value is reflected in the measured performance."*
Deliverable · kernel.py	*"The submission consists **only of /app/kernel.py**, which must be at most 300000 bytes."* *"You may cache compiled artifacts, static metadata, and output buffers. You **must not cache, precompute, or reuse any numerical result** that depends on a particular input."*
Deliverable · SKILL.md	*"It records **kernel knowledge only**: evidence-backed conclusions about how this computation maps efficiently to the target hardware, written so that they can **transfer to another kernel DSL**."* *"Every conclusion must rest on **concrete evidence**: a measured change in correctness, a time or speedup reported by the evaluator, or a specific observation about generated code or hardware behavior."* *"state the final conclusions **without relying on the current DSL's syntax or APIs**."* *"**Do not include** legality rules, code, commands, API instructions, compilation or debugging narratives, submission mechanics, or unsupported generic optimization advice."*
Implementation boundary	*"at least one returned output tensor must depend on a tensor value **computed or moved by a ⟨DSL⟩ kernel** launched during that call."* *"routing an already valid eager result through a semantics-preserving copy, cast, or layout change **does not satisfy the requirement**."* *"Do not invoke another kernel language or compiler, including torch.compile, torch.jit, Triton, or **raw CUDA C/C++**. None of these may be accessed through **inline-source facilities …**"* *"**CUDA Graphs are forbidden** because they remove launch overhead that the baseline still incurs."* *"A submission without a working ⟨DSL⟩ contribution is invalid and **receives a score of 0**, even if it is otherwise correct and fast."*
Grading contract	*"The maximum absolute error of your output relative to the anchor must not exceed **ALLOWANCE_FACTOR * max(reference_error, allowance_floor)** … An output more accurate than the original reference output always passes …"* *"After WARMUP_DRAWS warmup calls, grading takes the **median** over TIMING_DRAWS fresh inputs. The eager PyTorch baseline evaluates reference(…) using the **same warmup seeds, timing seeds, and measurement procedure**."* *"This **end-to-end** measurement includes Python dispatch within solve(…), all kernel launches, and all GPU work issued by eager PyTorch and ⟨DSL⟩."*
Environment	*"Your development sandbox provides 2 CPU cores and 8 GB of memory, but **no GPU or network access**. Grading runs in the same software image on one NVIDIA B200 (sm_100a), accessible only through the submission script below."* *"The ⟨DSL⟩ documentation is available under **/docs**. Inspect the directory first, then search it for specific APIs, architecture questions, or errors."*
Submissions and budget	*"You have **10 feedback submissions** in total. … After all submissions have been used, no further feedback submissions are accepted; the final evaluation described below still runs."* *"The total wall-clock budget is **40 minutes**, including time spent waiting for reports."* *"When the budget expires or you finish early, the version of /app/kernel.py **then present on disk** is evaluated again on a fresh B200; that evaluation **determines the final score**."*
Suggested workflow	*"Make the first version valid from the outset: implement the computation with eager PyTorch and add the **smallest genuine ⟨DSL⟩ kernel** whose result contributes to a returned output."* *"Submit a first candidate **before a quarter of the budget** has passed, even if it is a plain, unoptimized kernel: a correct slow kernel scores, an unfinished fast one does not."* *"After each report, add **only the newly acquired kernel knowledge and its evidence** to /app/SKILL.md. Do not copy reports or narrate the submission process."* *"Change **one key factor per experiment**, and state the expected effect before submitting."* *"During the last five minutes of the budget, stop high-risk experiments, perform the final checks, and **retain the best version**."*
Skill attachment	skill-conditioned attempts only · appended to the instruction; the source's kernel, trajectory, and DSL docs are withheld *"## Skill from a prior attempt"* *"This skill was distilled while working on **this same problem** in **a different kernel DSL**."* followed by the source attempt's SKILL.md, verbatim

Figure 8: The task instruction. The top row summarizes the template's parameters, and the cards quote excerpts from its sections in order, with ellipses marking omitted text. Bold emphasis is added.

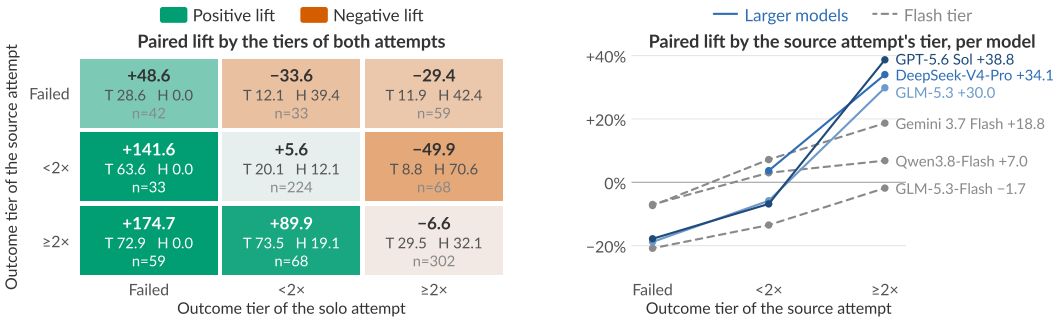


Figure 9: Paired lift by source and solo outcome tiers. **Left:** all 888 comparisons pooled over models and directions, with cells showing ΔGM, help rate T, and harm rate H in percent, and sample size n. **Right:** ΔGM by source tier for each model, pooling both directions. Passing tiers use speedup relative to the eager reference, and skipped treatments count as neutral.

B Additional Transfer Analyses

B.1 Outcome Tiers

Within each solo outcome tier, including the two where solo passes, sources at $2\times$ or better yield the highest paired lift, so the pattern does not rest on rescuing failed solo attempts (Figure 9, left). When solo already reaches $2\times$, even the highest source tier has negative lift. Pooled over solo tiers, lift rises with source tier in every model (Figure 9, right). From sources at $2\times$ or better, the three larger models gain 30.0% to 38.8%, against -1.7% to $+18.8\%$ for the Flash tier.

B.2 Direction Asymmetry and Source Tracking

When one direction is helped, the other is harmed more often than helped, which regression toward a problem's attainable speed cannot produce, since a retry with no skill involved would leave the two directions unrelated. Across the 444 model and problem pairs, the two directions' log paired ratios have a Spearman correlation of -0.26, which is negative within every model and family and reaches -0.34 among the 283 pairs whose four attempts all pass. In 20,000 draws from a reference distribution that independently flips the sign of each direction's log paired ratio with probability one half, preserving magnitudes and zeros, no correlation is as negative as the observed one.

Skill-conditioned attempts also track the source attempt's speed more closely than solo attempts do. Across the 841 comparisons with a transferred skill, the Spearman correlation with source speedup is 0.35 for solo attempts and 0.51 for skill-conditioned attempts. Speedups are floored at $0.05\times$, with nonpassing attempts assigned this floor before ranking. A reference distribution holds the source fixed and independently swaps the solo and skill-conditioned speedups within each comparison with probability one half, and none of its 2,000 draws yields a correlation gap at least as large as observed. Pass rates also follow the source verdict, with skill-conditioned attempts passing in 62.9% of the 97 comparisons with a nonpassing source against 75.3% for solo attempts on the same variants.

C Annotation Labels

C.1 Content Categories and Uptake Ratings

Each source skill receives a binary label for each of the 11 content categories of Table 3, and each skill-conditioned attempt an uptake rating and a propagation flag. A category is present when at least one statement in the skill matches its criterion. The labels describe a skill's claims and the evidence it states, without verifying either. An attempt's implemented design is rated followed fully when it mirrors the skill's guidance, followed partially when it follows some of it, ignored when the skill goes unused, contradicted when the attempt explicitly tests against or rejects a claim, and not applicable when the skill offers no actionable guidance. Propagation means that a false or unverified source claim reappears in the attempt's own skill, and an attempt without a skill of its own counts as

Table 3: Source-skill content categories. Each category is a binary label for a complete skill file.

Category	Presence criterion
Workload census	Enumerated workload properties beyond a restatement of the task.
Bottleneck and timing evidence	Measured bottleneck claims or timings tied to specific changes.
Verified reformulation	Implemented algorithmic or algebraic rewrites reported as correct.
Semantic and precision hazards	Specific semantic constraints or numerical accuracy boundaries.
Verification recipes	A correctness oracle, probe, or control reported as used.
Bounded failure diagnosis	Failure or slowdown evidence limited to what was tested.
Source-DSL mechanics	Guidance centered on the source DSL or its toolchain.
Ambition capping	Minimality or delegation advice without measured justification.
Unsupported closure	A conclusion that rules out alternatives beyond the stated evidence.
Agenda handoff	Optimization suggestions or hypotheses left open for later work.
Unmeasured prescriptions	Recommendations without supporting measurements or derivation.

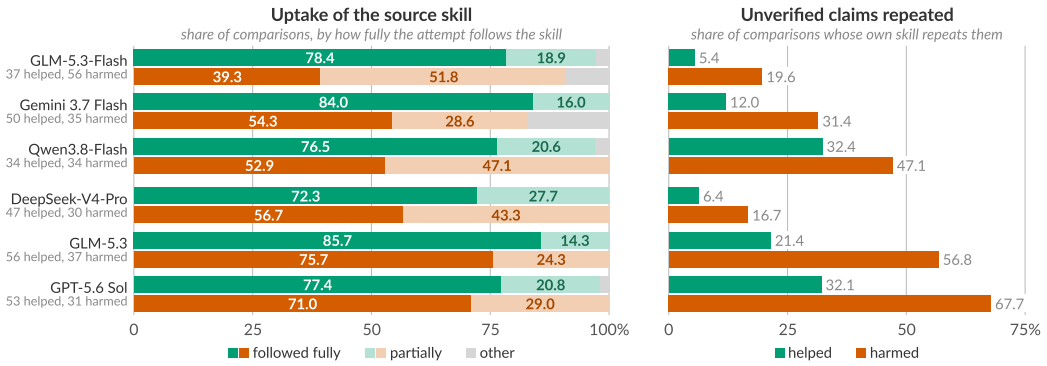


Figure 10: Skill uptake by transfer outcome. **Left:** how fully skill-conditioned attempts follow the source skill, with gray segments for uptake rated ignored, contradicted, or not applicable. **Right:** attempts whose own skill repeats unverified source claims. Shares are computed over each model's helped and harmed comparisons with a transferred skill, whose counts appear under the model name.

Table 4: Primary causes of harm. Each harmed comparison with a target-side failure receives one class based on the skill-conditioned attempt relative to solo on the same variant.

Class	Criterion
Search truncation	The source design limits search for better target-DSL options.
Unverified adoption	Untested reliance on a testable source claim that is wrong, unverified, or out of scope causes harm.
Faulty execution	The target's coding or debugging errors undermine a sound design.
Toolchain friction	Target-DSL API, compilation, or runtime barriers cause harm.
Budget and artifact mismanagement	Poor scheduling, final selection, or file handling leaves a worse or unusable implementation.
Rule and precision failure	Implementation rules are violated or tolerance-edge checks fail.

no propagation. Figure 10 gives the per-model uptake and propagation shares behind Section 4.4 by transfer outcome, with skills followed fully or partially in 82.9% to 100% of harmed and 97.1% to 100% of helped comparisons. The family panel of Figure 7 uses all comparisons with a transferred skill within each model and family, neutral ones included.

C.2 Failure Classes

Of the 223 harmed comparisons, 211 show a target-side failure and receive one class from Table 4, while the other 12 are excluded from the failure-cause counts in the left panel of Figure 7, whose denominator stays at 148 comparisons per model. When several classes apply, precedence runs from rule violation through faulty execution, budget and artifact mismanagement, toolchain friction,

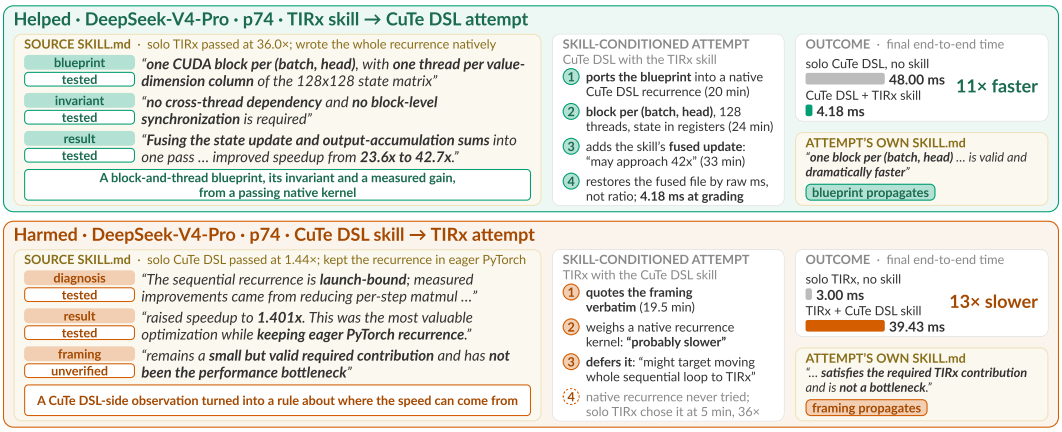


Figure 11: Bidirectional transfer on p74, the delta-rule recurrence, with DeepSeek-V4-Pro. **Top:** TIRx to CuTe DSL. **Bottom:** CuTe DSL to TIRx. Tags distinguish GPU-graded source evidence from unverified claims. Outcome panels report final, unclipped end-to-end runtimes against solo.

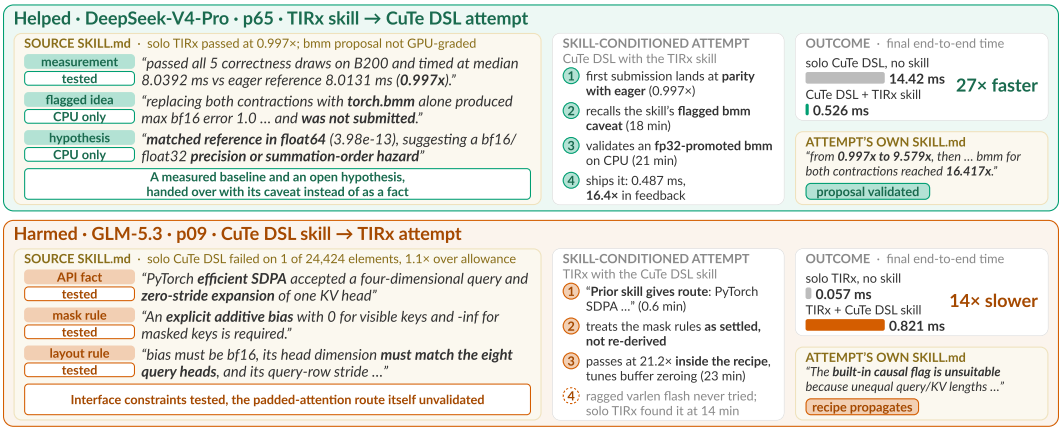


Figure 12: Hypothesis and recipe as transferred guidance. **Top:** DeepSeek-V4-Pro, TIRx to CuTe DSL on p65. **Bottom:** GLM-5.3, CuTe DSL to TIRx on p09. Tags distinguish GPU-graded evidence from CPU-only checks. Outcome panels report final, unclipped end-to-end runtimes against solo.

unverified adoption, and tolerance-edge loss to search truncation, which covers the remaining slowdowns. The 15 rule and precision failures of Gemini 3.7 Flash comprise 12 inline CUDA exclusions, two other rule violations, and one tolerance-edge loss.

D Additional Case Studies

The two cases in Section 4.3 come from passing GLM-5.3 sources, one per direction, and the four cases here cover both directions of one problem and sources at every relative standing. On p74, DeepSeek-V4-Pro's TIRx solo attempt passed at $36.0\times$ with a native recurrence and its CuTe DSL solo attempt at $1.44\times$ with an eager recurrence (Figure 11). As sources, the TIRx attempt stands above its target's solo attempt and the CuTe DSL attempt below. The CuTe DSL skill-conditioned attempt ports the blueprint and fused update into a native recurrence and runs faster than solo. The TIRx skill-conditioned attempt quotes the unverified framing that the DSL part is not the bottleneck, judges a native recurrence likely to be slower, never tries one, and runs slower than solo. With model and problem fixed, the outcome follows the source's standing, and each attempt's own skill carries forward what it received, the blueprint in one direction and the framing in the other.

Neither source in Figure 12 stands above the target's solo attempt, yet one transfer helps and the other does harm. The p65 TIRx source of DeepSeek-V4-Pro, which passed at $0.997\times$, records a

batched-matmul rewrite tested only on CPU as an open hypothesis with a precision caveat. The CuTe DSL skill-conditioned attempt validates a float32-promoted variant on CPU and then through GPU grading, runs faster than solo, and records the proposal as validated in its own skill. The p09 CuTe DSL source of GLM-5.3, which failed correctness, records tested scaled dot-product attention (SDPA) interface, mask, and layout rules as a route its own attempt had not validated. The TIRx skill-conditioned attempt adopts the route within its first minute, treats its rules as settled, never tries the variable-length flash attention that the solo attempt finds, runs slower than solo, and carries the rules into its own skill. In both cases the target treats the guidance as the source framed it, verifying the hypothesis before use and following the recipe without re-deriving it.

E Discussion

E.1 Limitations

The findings concern six models transferring skills between CuTe DSL and TIRx on a B200 under one agent scaffold, one skill directive, and one per-attempt budget. Source and skill-conditioned attempts come from the same model by design, so the findings do not separate how well a model distills its experience from how well it uses it in the other DSL. The content analysis relates transfer outcomes to the presence of content categories, not to individual statements or their wording. The same-problem design holds the relevance of prior experience fixed, so the findings do not cover generalization to new problems or workloads, nor the accumulation of skills over rounds.

E.2 Future Work

Comparing source implementations, trajectories, and skills could test how the choice of transferred material affects reuse, while comparing free-form and structured skills with matched content could test how organization affects uptake. Transferring skills between models could separate a model's ability to distill experience from its ability to use it and could test whether the source's relative standing also predicts transfer from larger models to Flash-tier models. Removing or revising individual guidance statements could test their effects on the target attempt's search, verification, and outcome. Transferring a skill to related problems in the same family could test which findings still apply when the problem changes and the agent must judge their relevance itself. Chaining transfer across rounds and DSLs could track whether useful knowledge is retained and refined, whether unverified claims are verified, corrected, or propagated, and how these changes affect subsequent attempts.