Verigrad: Verification-Driven Multi-Agent GPU Kernel Generation for High-Order MLIP Derivatives

Yao Liu[1,2] **Yuanchang Zhou**[1,2] **Hongtao Xu**[1,2] **Mingzhen Li**[1]*

[1]State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

[2]University of Chinese Academy of Sciences, Beijing, China

Abstract

Generating GPU kernels for derivatives requires preserving dependencies across forward, backward, and higher-order automatic differentiation phases. Local tensor checks can miss errors in the assembled scientific outputs. We present VERIGRAD, a verification-driven multi-agent system for first- and second-order derivative workloads in machine learning interatomic potentials (MLIPs). A `DerivativeTask` specifies requested quantities, tensor bindings, and checks; a derivative-aware intermediate representation exposes saved tensors and reconstruction rules to synthesis agents. An execution-based verifier accepts the composed derivative artifact, and a hardware-refinement agent retains lower-latency candidates only when they pass the same task checks. On KernelBench L3 double-backward targets, generation success is 78% for KernelAgent and 88% for VERIGRAD. With semantic lifting, reported time and token use per successful MatRIS operator are 33% and 34% lower, respectively. Integrated MatRIS inference and training on RTX 3090 achieve 1.33–$1.55\times$ speedup over eager PyTorch across twelve workloads.

1 Introduction

Forces, stresses, and response properties derived from learned energy functions are central to machine learning interatomic potentials (MLIPs) [Behler and Parrinello, 2007, Deringer et al., 2021, Musaelian et al., 2023]. Computing these quantities, or training against derivative targets, couples multiple automatic differentiation (AD) phases through saved tensors, sparse reductions, and reconstruction conventions. A locally correct kernel can still use the wrong saved tensor, force sign, atom axis, or second-order probe when integrated into the application.

LLM-based kernel agents automate synthesis, execution testing, repair, and performance tuning [Li et al., 2025b, PyTorch Team, 2025, 2026]. High-order derivatives expose a mismatch between the local programs these agents optimize and the composed quantities applications require. Our derivative-lift experiments on KernelBench [Ouyang et al., 2025] show decreasing generation reliability as targets move from forward computation to backward and double backward (Figure 1).

We identify three requirements: **C1: explicit derivative semantics**, including target roles, bindings, and cross-phase dependencies; **C2: composed-artifact acceptance**, including reconstruction of the requested outputs; and **C3: hardware tuning under the task contract**, revalidating candidates during GPU-specific optimization.

VERIGRAD addresses these requirements by making the derivative task the shared unit of synthesis, composition, verification, and refinement. It builds on the agentic execution substrate of isolated workers, typed tensor contracts, execution feedback, and profiling. Its contributions are:

*Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).



Figure 1: Generation success on KernelBench forward, backward, and double-backward targets. Bars show generation-success percentages for each derivative target and level.

1. **Derivative-task representation (C1).** `DerivativeTask` and a derivative-aware intermediate representation (IR) expose scientific roles, AD phases, saved-tensor provenance, and output reconstruction to synthesis agents.
2. **Task-level generation and repair (C2).** A verifier checks the composed derivative bundle against reference autograd and returns structured feedback for repair.
3. **Hardware-guided refinement (C3).** A profiling-guided agent tunes tiling, memory layout, and launch configurations for the target GPU. Candidates with lower measured latency replace accepted artifacts only after passing the same task checks.

We evaluate energy-induced first- and second-order MLIP derivatives at both operator and application levels.

2 Background and problem formulation

Derivative workloads. For scalar energy $E = f_\theta(x)$ at positions x, forces are $F = -\nabla_x E$. A Jacobian-vector product (JVP) applies a Jacobian to a tangent; a vector-Jacobian product (VJP) applies an output cotangent to a Jacobian. An energy Hessian-vector product (HVP) can be computed without materializing the Hessian:

$$h = Hv = \mathrm{JVP}(\nabla_x E; x, v), \qquad \mathrm{JVP}(F; x, v) = -h. \tag{1}$$

Force-target training also differentiates with respect to parameters. For a loss $\ell(E, F)$, the chain rule gives

$$\nabla_\theta \ell = (\nabla_\theta E)\, \partial_E \ell - \left(\frac{\partial(\nabla_x E)}{\partial \theta}\right)^\top \nabla_F \ell. \tag{2}$$

The mixed position–parameter derivative in Equation 2 differs from the position HVP in Equation 1. A training task must therefore include the required loss cotangent and parameter-backward action; an energy/force/HVP output bundle alone does not specify force training.

Task and acceptance. Given a differentiable model and representative tracing inputs X_{ex}, the objective is a lower-latency GPU artifact A_T for the task

$$T = (\mathcal{Y}, \mathcal{D}, \mathcal{B}, \varepsilon, \mathcal{C}),$$

where $\mathcal{Y}$ names requested quantities, $\mathcal{D}$ specifies derivative modes, $\mathcal{B}$ binds logical objects to tensors, $\varepsilon = \{(\mathtt{atol}_y, \mathtt{rtol}_y) : y \in \mathcal{Y}\}$ gives per-output tolerances, and $\mathcal{C}$ lists executable checks with declared preconditions. For reference bundle R_T, acceptance requires

$$\mathrm{Pass}_T(A_T) = \bigwedge_{x \in X_{\mathrm{ver}}} \mathrm{VERIFY}(A_T(x), R_T(x), T).\mathrm{pass}. \tag{3}$$

The verifier evaluates the task on selected inputs X_{ver} using a stable reference execution.

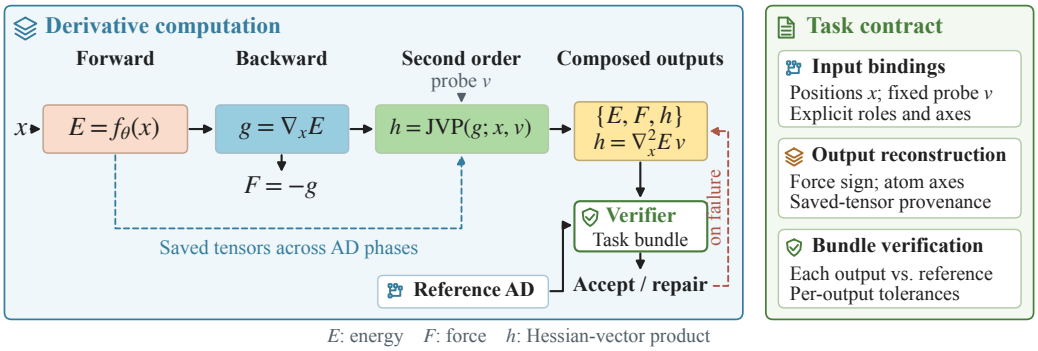


Figure 2: Energy/force/HVP computation under a task contract. AD units share saved tensors; the verifier compares the assembled $\{E, F, h\}$ with reference AD under per-output tolerances and returns repair feedback on failure.

Worked example: energy, force, and HVP. A task requests $\{E, F, h\}$ with $F = -\nabla_x E$ and $h = \nabla_x^2 E\, v$. Its IR separates energy evaluation, position gradients, and HVP propagation through saved tensors (Figure 2). Reconstruction fixes signs, axes, and the probe binding. Figure 4 contrasts locally correct units with incorrect compositions; Appendix B gives the descriptor.

3 System design

VERIGRAD follows a lift–synthesize–verify–refine pipeline (Figure 3). Algorithm 1 summarizes its acceptance and replacement rules: generation seeks a passing artifact, while refinement seeks a lower-latency artifact that still passes.

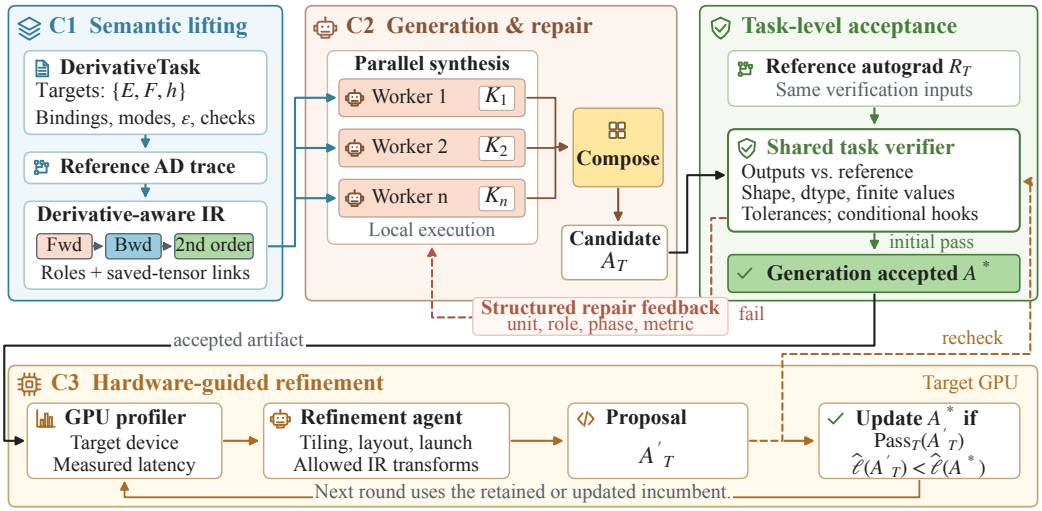


Figure 3: VERIGRAD architecture. C1 supplies derivative semantics to parallel synthesis agents (C2); composed candidates receive task-level acceptance or structured repair feedback. C3 tunes accepted artifacts on the target GPU. Proposals return to the same verifier and replace the incumbent only after passing and reducing measured latency.

3.1 Derivative tasks as explicit generation inputs

Semantic lifting. The user specifies targets, derivative modes, logical bindings, tolerances, and optional checks through `DerivativeTask`. Reference AD tracing on X_{ex} resolves objects such as

`positions`, `cell`, `energy`, and `probe_v` to concrete tensors and derivative paths. Resolution records batching, atom/edge axes, signs, periodic-cell metadata, and reduction directions; ambiguous or inconsistent bindings are rejected. The descriptor and trace jointly supply the semantics exposed to synthesis, rather than requiring workers to infer scientific roles from anonymous tensor programs.

IR and lowering. Lowering partitions the derivative program at AD phase boundaries, saved-tensor dependencies, output reductions, and layout-sensitive sparse operations such as neighbor gather/scatter. Each IR unit carries an `id`, a `phase` (forward/backward/double backward), a `role` (energy/force/stress/HVP path), typed bindings, `SavedTensorRef` provenance, cross-phase edges, reconstruction rules, verifier hooks, and allowed transforms. Specialization keys record dtype, device, rank, symbolic shapes, neighbor layout, and boundary metadata. Local oracles compare captured tensor counterparts; composition supplies the checks that depend on cross-phase reuse and reconstruction. An IR unit need not map one-to-one to a GPU launch.

3.2 Verifier-gated generation and repair

Composition. Isolated workers synthesize kernels for IR units, using local execution feedback where available. Composition specifies tensor materialization, saved-tensor reuse, permitted buffer aliasing, and assembly of requested quantities. The resulting artifact can replace a captured derivative subprogram or be integrated through a custom autograd-compatible function. For training, this integration must expose the parameter-backward action required by the loss, not just return forward derivative quantities.

Task-level checks. The acceptance gate executes the reconstructed artifact and R_T on the same inputs. Numerical checks require matching shapes and dtypes, finite reference and candidate values, and agreement for every tensor element

$$|y_{\text{cand}} - y_{\text{ref}}| \le \texttt{atol}_y + \texttt{rtol}_y |y_{\text{ref}}|.$$

Tolerances are specified per output before candidate evaluation. Physical hooks run only when their preconditions hold (Appendix B). This end-to-end comparison can reveal the output effects of saved-tensor misuse, aliasing, wrong axes, signs, or probes on the tested inputs. Provenance metadata supports attribution of output mismatches to their producing units.

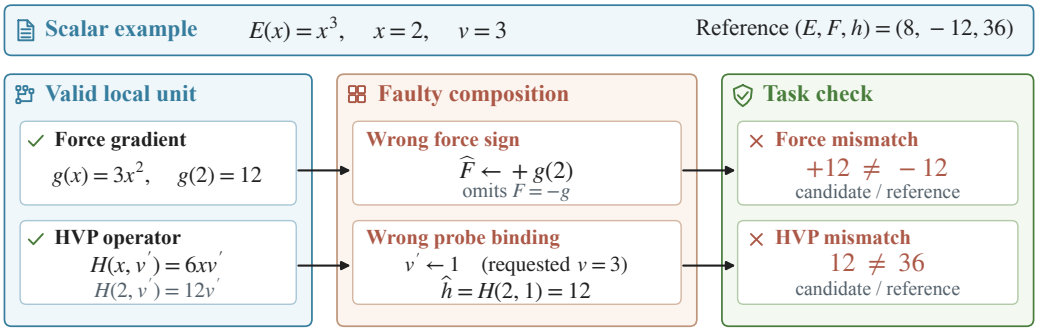


Figure 4: Local correctness can fail task acceptance. For $E = x^3$, $x = 2$, and $v = 3$, the requested bundle is $(8, -12, 36)$. Incorrect force signs and probe bindings yield invalid composed outputs.

Structured repair. Rejection reports the implicated unit, role, AD phase, verifier layer, hook, and metric. This directs repair toward a scientific mismatch, such as force reconstruction or HVP probe binding, instead of an unnamed tensor error. Compilation/execution errors reject the candidate; unstable references or incompatible checks reject the task rather than widening tolerances. Repair changes a candidate to achieve acceptance; refinement tunes an already accepted candidate.

3.3 Verification-preserving hardware refinement

The refinement agent uses profiling feedback to propose tiling, memory layout, fusion, vectorization, staging, and launch changes according to the IR's allowed-transform policy. Derivative roles, bindings,

Algorithm 1 Task-gated generation and hardware refinement

Require: Task T, reference R_T, tracing inputs X_{ex}, verification inputs X_{ver}, generation/refinement budgets

$I \leftarrow \text{Lift}(T, \text{Trace}(R_T, X_{\text{ex}}))$; $A^* \leftarrow \emptyset$; $r \leftarrow \emptyset$
for generation rounds within budget **do**
  $K \leftarrow \text{SynthesizeOrRepair}(I, r)$ using available local oracles
  $A \leftarrow \text{Compose}(I, K)$; $r \leftarrow \text{Verify}(A, R_T, T, X_{\text{ver}})$
  if r.pass **then**
    $A^* \leftarrow A$; **break**
  end if
end for
if $A^* = \emptyset$ **then**
  return generation failure
end if
$p^* \leftarrow \text{Profile}(A^*)$; $\ell^* \leftarrow p^*$.latency
for refinement proposals within budget **do**
  $A' \leftarrow \text{Refine}(A^*, I, p^*)$ within allowed transforms
  if $\text{Verify}(A', R_T, T, X_{\text{ver}})$.pass **then**
    $p' \leftarrow \text{Profile}(A')$
    if p'.latency $< \ell^*$ **then**
      $(A^*, p^*, \ell^*) \leftarrow (A', p', p'.\text{latency})$
    end if
  end if
end for
return A^*

reconstruction, and saved-tensor semantics remain the contract. Writing $\widehat{\ell}(A)$ for measured latency, replacement obeys

$$A^*_{k+1} = \begin{cases} A'_k, & \text{Pass}_T(A'_k) \ \wedge \ \widehat{\ell}(A'_k) < \widehat{\ell}(A^*_k), \\ A^*_k, & \text{otherwise.} \end{cases} \tag{4}$$

A failed or slower proposal retains the accepted artifact; a passing, faster proposal updates both the incumbent and its profile.

4 Evaluation

We ask whether derivative-aware generation improves reliability (RQ1), which components explain the improvement (RQ2), whether refinement helps across GPUs (RQ3), and whether generated kernels accelerate integrated MLIP workloads (RQ4). RQ5 measures generation cost.

4.1 Setup and statistical protocol

Workloads and platforms. KernelBench derivative lifts cover L1/L2/L3. Application measurements use MatRIS-10M-OAM and MatRIS-10M-MP, each with energy (`e`), energy–force (`ef`), and energy–force–stress–magnetic-moment (`efsm`) bundles in inference and AdamW training: twelve workloads in total. Additional application results use MACE-style and NequIP/Allegro-style models.

The primary application platform is an RTX 3090 with 24 GB memory, PyTorch 2.4.0, and CUDA 12.4. Batch size and supercell size are both 1. Latencies average eight repetitions after three warmups. Cross-hardware refinement uses the MatRIS operator suite on RTX 3090, A100, A800, and H100.

Agents and measurement units. VERIGRAD and the matched KernelAgent comparison use `GPT-5.4 xhigh`. The component ablation reports a matched backend, generation budget, and repair cap; AutoTriton provides an additional derivative-lift baseline. Runtime baselines are eager PyTorch and `torch.compile`/TorchInductor. KernelBench measures derivative-target generation, MatRIS measures operator generation, and application timing measures integrated execution.

Statistical reporting. Component-ablation and MatRIS success/round summaries use mean ± standard deviation over three seeds. Cross-device plots summarize suite minima and maxima. Appendix A details the measurement protocols and statistical records.

4.2 RQ1: Reliability on derivative targets

Figure 1 compares forward, backward, and double-backward targets. The reported AutoTriton and KernelAgent success rates decrease with derivative order, particularly on L3. For double backward, VERIGRAD's reported rates exceed KernelAgent's by 2, 10, and 10 percentage points on L1, L2, and L3.

4.3 RQ2: Contributions of semantic lifting and verification

The sequential L3 double-backward ablation incrementally adds semantic lifting, composed-task verification, and structured repair feedback (Figure 5). Adding semantic lifting is associated with a 6 percentage-point increase in the reported mean, task-level verification with a further 2, and structured feedback with a further 2. C3 is evaluated separately because it starts after a candidate passes verification.

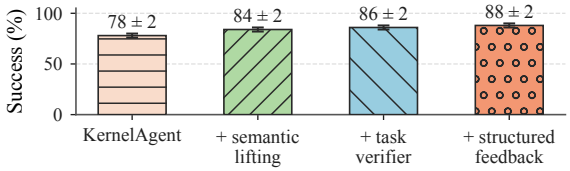


Figure 5: KernelBench L3 double-backward ablation (mean ± SD, three seeds). Components are added cumulatively.

The MatRIS operator ablation provides application-specific evidence: success with semantic lifting is $88 \pm 2\%$ versus $72 \pm 3\%$ without it; reported interaction rounds are 2.48 ± 0.15 versus 4.20 ± 0.34 (Figure 10). The 41% reduction in reported rounds is accompanied by lower time and token use.

4.4 RQ3: Hardware-guided refinement

Figure 6 separates the first verifier-passing artifact from the fastest accepted artifact after refinement. Across four GPUs, reported post-refinement operator speedups range from $1.21\times$ to $6.85\times$ over eager. Both endpoints of the speedup envelope are higher after refinement on each GPU.

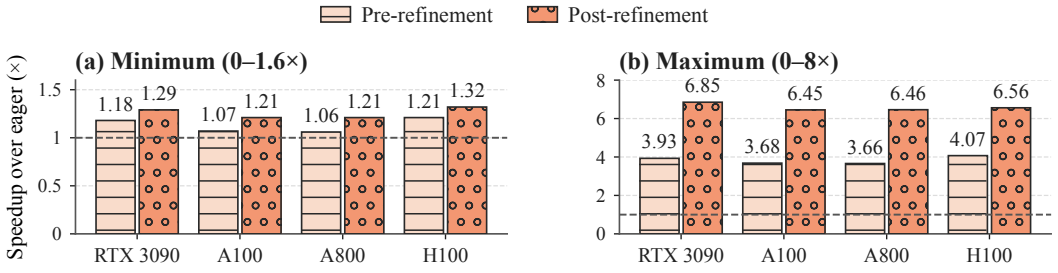


Figure 6: Suite minimum (a) and maximum (b) speedups over eager on four GPUs, before and after refinement (MatRIS operators, three seeds). Bars start at zero; panel scales differ. Dashed lines mark the $1\times$ eager baseline. Bars show unpaired suite extrema.

Figure 7 gives eight RTX 3090 operator latency records, separating eager, compiled, initial generated, and optimized execution. Appendix C discusses hardware profiling.

4.5 RQ4: Integrated MLIP performance

Figure 8 reports the twelve MatRIS workloads. VERIGRAD achieves 1.33–$1.55\times$ speedup over eager on RTX 3090. `torch.compile` has lower reported mean latency on energy-only inference for both OAM and MP; VERIGRAD has lower reported mean latency on the remaining ten plotted workloads.

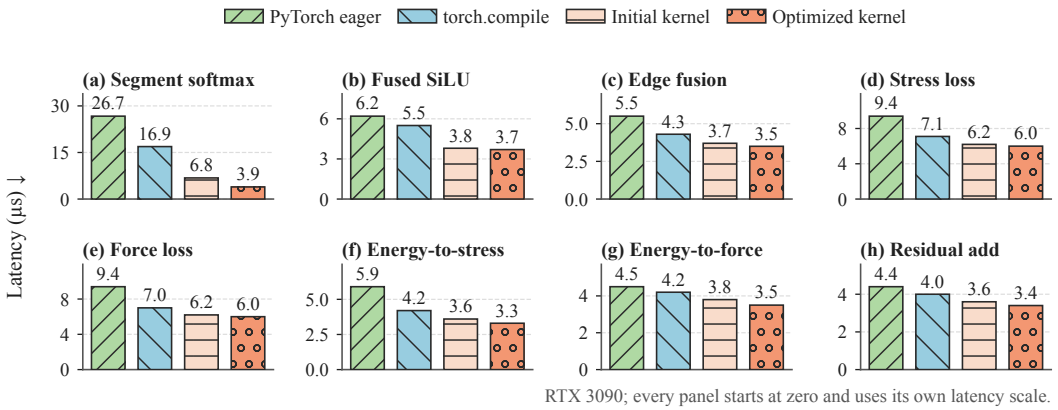


Figure 7: Eight RTX 3090 operator latency records (lower is better). Each panel compares eager, compiled, initial generated, and optimized execution on its own zero-based scale in μs. Panels share the method ordering and color scheme.

Timings include execution of the assembled workload, including launch and composition effects; offline LLM synthesis and refinement costs are reported separately.

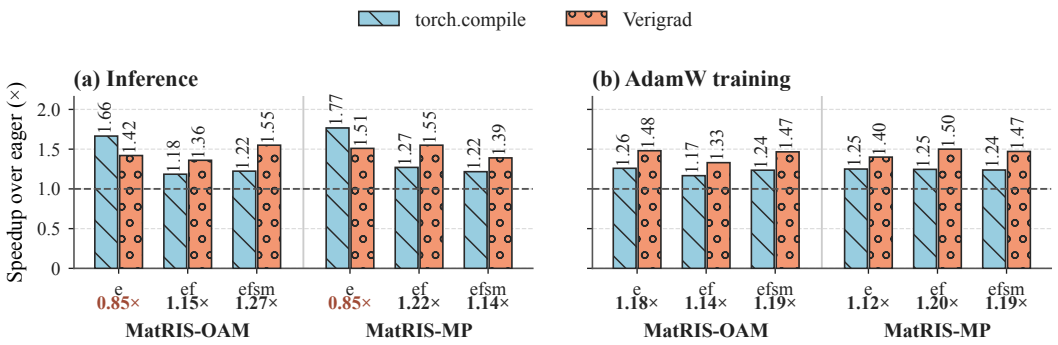


Figure 8: End-to-end MatRIS speedups on RTX 3090. Bars and their labels give speedup over eager (dashed $1\times$ line); bold ratios below the workload labels give VERIGRAD speedup over `torch.compile`. Ratios below $1\times$ indicate slower execution. Both panels use `e`/`ef`/`efsm`.

Two additional MACE-style and NequIP/Allegro-style configurations show approximately $1.44\times$ speedup over eager and 1.20–$1.22\times$ over `torch.compile` for full-output inference (Figure 9; Appendix D).

4.6 RQ5: Generation cost

The MatRIS semantic-lifting ablation jointly reports success, interaction rounds, wall-clock time, and token use (Figure 10). With semantic lifting, success rises by 16 percentage points, while mean rounds decrease from 4.20 to 2.48. Time per successful operator decreases from 467 to 312 seconds (33.2%), and token use from 73.2K to 48.6K (33.6%). Appendix E develops the cost and amortization model.

5 Related work

Scientific AD and MLIP systems. SchNet, NequIP, Allegro, MACE, and MatRIS derive forces and responses from learned energies [Schütt et al., 2017, Batzner et al., 2022, Musaelian et al., 2023, Batatia et al., 2022, Zhou et al., 2026]; universal atomistic models broaden these workloads [Zhang et al., 2025, Wood et al., 2025]. PyTorch/`torch.func`, JAX, Enzyme, and Dex differentiate programs or IRs [Paszke et al., 2019, PyTorch Team, 2023a, Bradbury et al., 2018, Moses et al.,

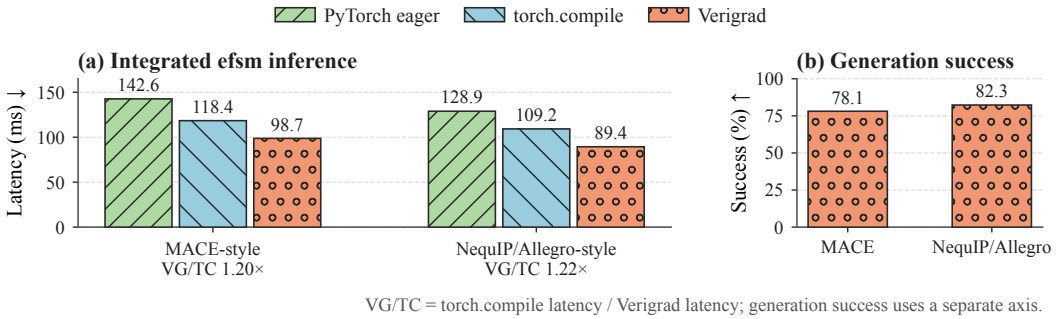


Figure 9: `efsm` inference for two architecture-style configurations on RTX 3090. (a) Latency compares eager, compiled, and VERIGRAD execution; (b) independently measured VERIGRAD generation success.

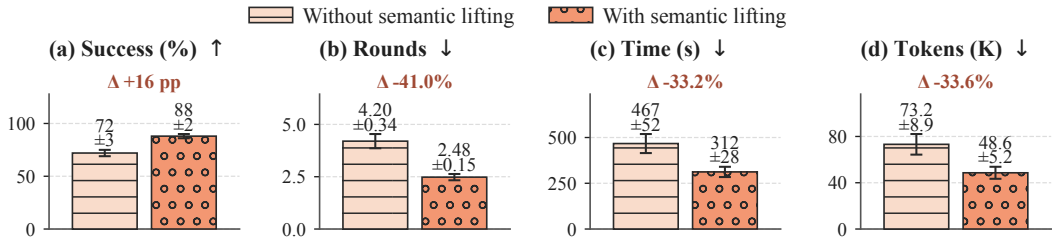


Figure 10: Semantic-lifting ablation on MatRIS operators. Bars show means and supplied dispersion. Success/rounds use standard deviations over three seeds; rounds, time, and tokens are per successful operator (Appendix A).

2021, Paszke et al., 2021]. VERIGRAD uses reference AD and carries the requested scientific outputs through synthesis and acceptance.

Compilers and kernel agents. Triton, MLIR, TorchInductor, and XLA provide compilation substrates [Tillet et al., 2019, Lattner et al., 2020, PyTorch Team, 2023b, XLA Team, 2017]; KernelBench and TritonBench provide benchmarks [Ouyang et al., 2025, Li et al., 2025a]. AutoTriton, GEAK, KernelFalcon, and KernelAgent explore LLM-driven kernel optimization [Li et al., 2025b, Wang et al., 2025, PyTorch Team, 2025, 2026, Meta PyTorch, 2025]. Our derivative-task contract extends this substrate with scientific roles, cross-phase dependencies, and composed-artifact acceptance.

Formal kernel verification. ProofWright combines agentic CUDA generation with formal safety and semantic checks [Chatterjee et al., 2025]; VOLTA checks equivalence of supported ML GPU kernels [Driscoll et al., 2025]. VERIGRAD complements unit-level formal checks with executable task contracts for scientific roles and reconstruction across units.

6 Discussion and limitations

Reference and input coverage. Acceptance depends on reference quality and verification-input coverage. Repeated feedback can adapt candidates to those inputs, so changes in shapes, sparsity, boundaries, or cutoffs require revalidation. The evaluation covers energy-induced first- and second-order MLIP derivatives with fixed execution structure; dynamic neighbor construction and broader derivative programs require additional contracts and tests.

Hardware and resource coverage. Operator results span four NVIDIA GPUs, and application measurements use RTX 3090. Extending the workflow to new backends requires compiler support and numerical/profiling checks. Distributed derivatives additionally require contracts for owned/ghost atoms, halo exchange, reductions, and communication ordering. Paired timing records and campaign-level device accounting would support finer performance and resource analysis.

Broader impact. Efficient derivatives can reduce scientific-computing cost, while undetected errors can corrupt forces or responses. Generation also consumes compute. Task-scoped checks and offline cost reporting help assess these tradeoffs; deployment requires validation for its numerical and physical conditions.

7 Conclusion

VERIGRAD connects kernel synthesis, composition, and hardware refinement through an explicit derivative-task contract. Evaluation shows higher generation success with semantic lifting and lower per-success generation cost; integrated MatRIS workloads achieve 1.33–$1.55\times$ speedup over eager.

Acknowledgments and Disclosure of Funding

This work is supported by the following funding: the Strategic Priority Research Program of Chinese Academy of Sciences (XDA0480504), National Science Foundation of China (62502501), National Key Research and Development Program of China (2025YFB4507000), Beijing Natural Science Foundation (4254087), and the Innovation Funding of ICT, CAS. Part of the training is performed on the robotic AI-Scientist platform of Chinese Academy of Sciences. The authors declare no competing interests.

References

Ilyes Batatia, David P Kovacs, Gregor Simm, Christoph Ortner, and Gábor Csányi. Mace: Higher order equivariant message passing neural networks for fast and accurate force fields. *Advances in neural information processing systems*, 35:11423–11436, 2022.

Simon Batzner, Albert Musaelian, Lixin Sun, Mario Geiger, Jonathan P Mailoa, Mordechai Kornbluth, Nicola Molinari, Tess E Smidt, and Boris Kozinsky. E (3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials. *Nature communications*, 13(1):2453, 2022.

Jörg Behler and Michele Parrinello. Generalized neural-network representation of high-dimensional potential-energy surfaces. *Physical review letters*, 98(14):146401, 2007.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs. `http://github.com/google/jax`, 2018.

Bodhisatwa Chatterjee, Drew Zagieboylo, Sana Damani, Siva Hari, and Christos Kozyrakis. ProofWright: Towards agentic formal verification of CUDA. *arXiv preprint arXiv:2511.12294*, 2025. URL `https://arxiv.org/abs/2511.12294`.

Volker L Deringer, Albert P Bartók, Noam Bernstein, David M Wilkins, Michele Ceriotti, and Gábor Csányi. Gaussian process regression for materials and molecules. *Chemical reviews*, 121(16): 10073–10141, 2021.

Benjamin Driscoll, Kshitij Dubey, Anjiang Wei, Neeraj Kayal, Rahul Sharma, and Alex Aiken. Equivalence checking of ML GPU kernels. *arXiv preprint arXiv:2511.12638*, 2025. URL `https://arxiv.org/abs/2511.12638`.

Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. Mlir: A compiler infrastructure for the end of moore's law. *arXiv preprint arXiv:2002.11054*, 2020.

Jianling Li, Shangzhan Li, Zhenye Gao, Qi Shi, Yuxuan Li, Zefan Wang, Jiacheng Huang, WangHaojie WangHaojie, Jianrong Wang, Xu Han, et al. Tritonbench: Benchmarking large language model capabilities for generating triton operators. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 23053–23066, 2025a.

Shangzhan Li, Zefan Wang, Ye He, Yuxuan Li, Qi Shi, Jianling Li, Yonggang Hu, Wanxiang Che, Xu Han, Zhiyuan Liu, et al. Autotriton: Automatic triton programming with reinforcement learning in llms. *arXiv preprint arXiv:2507.05687*, 2025b.

Meta PyTorch. KernelAgent: Multi-agent GPU kernel synthesis and optimization. `https://github.com/meta-pytorch/KernelAgent`, 2025. GitHub repository.

William S. Moses, Valentin Churavy, Ludger Paehler, Jan Hückelheim, Sri Hari Krishna Narayanan, Michel Schanen, and Johannes Doerfert. Reverse-mode automatic differentiation and optimization of GPU kernels via Enzyme. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2021.

Albert Musaelian, Simon Batzner, Anders Johansson, Lixin Sun, Cameron J Owen, Mordechai Kornbluth, and Boris Kozinsky. Learning local equivariant representations for large-scale atomistic dynamics. *Nature Communications*, 14(1):579, 2023.

Anne Ouyang, Simon Guo, Simran Arora, Alex L Zhang, William Hu, Christopher Ré, and Azalia Mirhoseini. Kernelbench: Can llms write efficient gpu kernels? *arXiv preprint arXiv:2502.10517*, 2025.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

Adam Paszke, Daniel D. Johnson, David Duvenaud, Dimitrios Vytiniotis, Alexey Radul, Matthew J. Johnson, Jonathan Ragan-Kelley, and Dougal Maclaurin. Getting to the point: Index sets and parallelism-preserving autodiff for pointful array programming. In *Proceedings of the ACM on Programming Languages (ICFP)*, 2021.

PyTorch Team. torch.func: Composable function transforms for PyTorch. `https://pytorch.org/docs/stable/func.html`, 2023a. PyTorch documentation.

PyTorch Team. `torch.compile` / TorchInductor official documentation. `https://pytorch.org/docs/stable/torch.compiler.html`, 2023b.

PyTorch Team. KernelFalcon: Autonomous GPU kernel generation via deep agents. PyTorch blog, 2025. PyTorch blog.

PyTorch Team. KernelAgent: Hardware-guided GPU kernel optimization via multi-agent orchestration. PyTorch blog, 2026. PyTorch blog.

Kristof Schütt, Pieter-Jan Kindermans, Huziel Enoc Sauceda Felix, Stefan Chmiela, Alexandre Tkatchenko, and Klaus-Robert Müller. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. *Advances in neural information processing systems*, 30, 2017.

Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 10–19, 2019.

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.

Brandon M Wood, Misko Dzamba, Xiang Fu, Meng Gao, Muhammed Shuaibi, Luis Barroso-Luque, Kareem Abdelmaqsoud, et al. UMA: A family of universal models for atoms. *arXiv preprint arXiv:2506.23971*, 2025.

XLA Team. XLA: Optimizing compiler for machine learning. `https://www.tensorflow.org/xla`, 2017.

Duo Zhang, Anyang Peng, Chun Cai, Wentao Li, Yuanchang Zhou, Jinzhe Zeng, Mingyu Guo, Chengqian Zhang, Bowen Li, Hong Jiang, Tong Zhu, Weile Jia, Linfeng Zhang, and Han Wang. A graph neural network for the era of large atomistic models. *arXiv preprint arXiv:2506.01686*, 2025.

Yuanchang Zhou, Siyu Hu, Xiangyu Zhang, Hongyu Wang, Guangming Tan, and Weile Jia. MatRIS: Toward reliable and efficient pretrained machine learning interatomic potentials. *arXiv preprint arXiv:2603.02002*, 2026.

A Evaluation protocol and statistics

Generation success counts accepted artifacts among eligible attempts; application speedup is reference latency divided by candidate latency. Table 1 summarizes the experimental records and protocol fields.

Table 1: Experimental records and protocol fields.

Experiment	Retained evidence	Unavailable fields
KernelBench lifts	Endpoint success aggregates	Eligible task IDs/denominators, baseline revisions, seed IDs, exact budgets, common final gate
Component ablation	Four means and standard deviations; three seeds; matched-budget statement	Per-seed outcomes, precise cap, task list, common final evaluator
MatRIS generation	operators; success, rounds, time, tokens	Per-run status/cost, cost aggregation hierarchy, repair/refinement stage separation
Four-GPU refinement	Devices, three seeds, pre/post ranges	Device variants/software, paired operator samples, timing dispersion, profiling counters
Application latency	RTX 3090 stack; warmups/repetitions; speedups and selected latencies	Input structures, synchronization, individual samples, common-success agent coverage
Additional coverage	Two architecture-style configurations	Model/checkpoint definitions, target mappings, robustness/backend sweeps, campaign GPU-hours

Statistics. Component-ablation and MatRIS success/round statistics report mean $\pm$ standard deviation over three seeds. Figure 1 reports endpoint percentages. Time/token whiskers retain the supplied dispersion; its aggregation hierarchy and repeated-run application-latency dispersion are unspecified in the source records. Error bars therefore retain their metric-specific definitions.

Table 2: Numerical values underlying Figure 5: reported mean $\pm$ standard deviation over three seeds. Components are added sequentially under a matched backend and budget.

Setting	Semantic IR	Task verifier	Feedback	Success (%)
KernelAgent	no	no	no	78 ± 2
+ semantic lifting	yes	no	no	84 ± 2
+ task-level verifier	yes	yes	no	86 ± 2
Full generation	yes	yes	yes	88 ± 2

Timing and resource units. The recorded application protocol is three warmups followed by eight averaged repetitions, batch size 1, and supercell size 1. GPU-hours require device-time accounting across concurrent workers; they cannot be recovered from elapsed time that includes LLM API waits.

B Task contracts and verification hooks

Illustrative descriptor for Figure 2.
Tolerance pairs use (`atol`, `rtol`).

```
DerivativeTask(
    targets=["energy", "force", "hvp"],
    modes={
        "force": grad("energy", wrt="positions", sign="negative"),
        "hvp": hvp("energy", wrt="positions", vector="probe_v"),
    },
    bindings={
        "positions": "batch.pos",
        "energy": "model(batch).energy",
        "probe_v": "task.probe_v",
    },
    tolerance={
        "energy": (1e-4, 1e-4),
        "force": (5e-4, 5e-4),
        "hvp": (1e-3, 1e-3),
    },
    verifier_hooks=[
        "energy_force_consistency",
        "hvp_probe_binding",
        "output_reconstruction",
    ],
)
```

Table 3: Terminology connecting the user request, synthesis units, and acceptance gate.

Term	Meaning
`DerivativeTask`	Scientific output request, derivative modes, logical bindings, tolerances, and enabled checks.
IR unit	A typed synthesis unit with an AD phase, scientific role, dependencies, and reconstruction contract; it need not map one-to-one to a GPU launch.
AD phase	Forward, backward, or second-order differentiation stage.
Saved tensor	An intermediate whose value and provenance must remain consistent across phases.
Reconstruction rule	The mapping from unit outputs to requested quantities, including signs, axes, and reductions.
Verifier hook	A task-enabled executable consistency check with declared preconditions.
Artifact	The composed executable replacement, including generated kernels and their integration.
Repair / refinement	Changes to achieve verification acceptance / tuning of an already accepted artifact.

Table 4: Optional verification hooks, their preconditions, and possible diagnoses.

Hook	Preconditions	Typical diagnosis
Force–energy consistency	Differentiable scalar energy and declared position binding	Sign convention, missing reduction, or wrong position axis
Hessian/HVP symmetry	Smooth scalar energy and compatible probe vectors	Transposed Hessian block or incorrect second-order path
Translation force sum	Translationally invariant energy and compatible boundary convention	Coordinate convention or periodic displacement error
Stress/virial consistency	Periodic cell, declared cell perturbation rule, and virial convention	Stress reconstruction or cell-gradient mismatch
Equivariance/rotation check	Model and task declare the corresponding symmetry	Frame convention, tensor-basis, or output-axis error

The hook schemas can be made concrete through residual predicates. For example, force consistency measures $\|F + \nabla_x E\|_\infty$, translation consistency measures $\|\sum_i F_i\|_\infty$, and a smooth-energy HVP symmetry check compares two probes:

$$|u^\top h(v) - v^\top h(u)| \le a_{\text{hook}} + r_{\text{hook}} \max\{|u^\top h(v)|, |v^\top h(u)|\}. \tag{5}$$

Here $h(w) = \nabla_x^2 E\, w$ and hook tolerances are declared with the check. Enabled hooks and their preconditions belong to the task contract and supplement reference-output agreement.

C Operator refinement details

Figure 7 compares eager, compiled, initial generated, and optimized execution for eight operators on RTX 3090.

Hardware-limit context. Hardware profiling relates kernel latency to operation count, byte traffic, and sustained device bandwidth/throughput. DRAM and SM SOL metrics, occupancy, and bottleneck classification help distinguish memory, compute, and launch overhead when selecting a refinement transform.

D Application baselines and model families

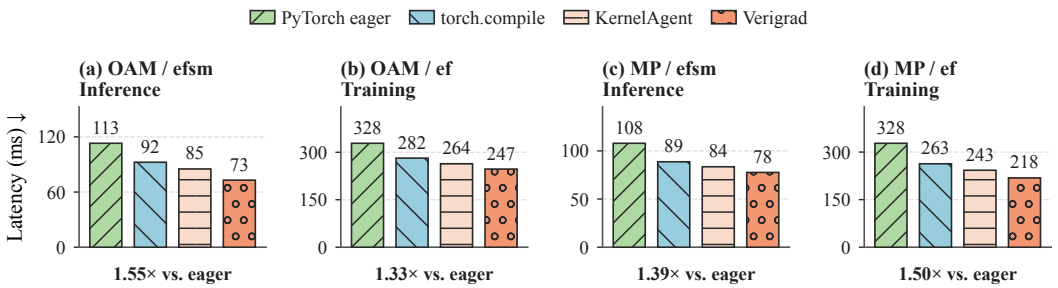


Figure 11: Absolute latency for four MatRIS workloads. KernelAgent measurements use local-oracle checks and finite-output admission (Appendix A).

The figure reports application latency under each method's admission rule. The KernelAgent records use local-oracle checks, while VERIGRAD checks the composed derivative task.

The cross-family results in Figure 9 use the RTX 3090 protocol. "MACE-style" and "NequIP/Allegro-style" identify the evaluated architecture configurations; `efsm` groups energy, force, stress, and magnetic-moment outputs.

E Generation cost and amortization

Figure 10 reports conditional cost per successful operator, including lifting, synthesis, verification, and refinement. Tokens aggregate generation and repair exchanges. Interaction rounds aggregate the agent exchanges. Campaign-level cost adds failed attempts and shared overhead to successful-case costs.

For a single repeatedly invoked workload, let t_{offline} be the incremental one-time generation/refinement time and t_{ref}, t_A its reference and generated execution latencies. An idealized serialized break-even condition is

$$N(t_{\text{ref}} - t_A) > t_{\text{offline}}, \qquad t_A < t_{\text{ref}}.$$

The break-even count depends on the complete offline investment and the latency saved on each invocation.

Additional input and backend coverage. Input-coverage extensions can vary atom/batch size, neighbor density/order, boundary conventions, and numerical edge cases under fixed tolerances. Generator-backend comparisons can use the same task contracts and verification inputs.

F Derivative operators and scientific output conventions

This section expands the derivative identities in Section 2. Sections G–L provide exact examples, interface schemas, and consequences of the stated acceptance rule. The interface schemas specify task contracts and information flow. Appendix M collects the existing numerical evidence.

F.1 Shapes and directional actions

Let $x \in \mathbb{R}^{N\times 3}$ denote positions and $q = \mathrm{vec}(x) \in \mathbb{R}^d$, where $d = 3N$. Flattening is a notation convention; implementations can retain atom and Cartesian axes. For scalar total energy $E_\theta(q)$,

$$g = \nabla_q E_\theta, \qquad F = -g, \qquad H = \nabla_q^2 E_\theta, \qquad h(q, v) = H(q)v. \tag{6}$$

The Hessian is the mathematical linear map underlying a position-based second-order operator. Computing its action does not require storing its $d \times d$ entries. For a smooth scalar energy, $H = H^\top$, so $\mathrm{JVP}(g; q, v)$ and $\nabla_q(g^\top v)$ both produce Hv when the probe is held fixed. A dense Hessian has d^2 entries; one HVP returns d entries, although intermediate AD storage still depends on the model.

For $u : \mathbb{R}^d \to \mathbb{R}^m$ with Jacobian J_u,

$$\mathrm{JVP}(u; q, v) = J_u(q)v, \qquad \mathrm{VJP}(u; q, w) = J_u(q)^\top w. \tag{7}$$

The tangent v has the input shape and cotangent w the output shape. For forces, $J_F = -H$, hence $\mathrm{JVP}(F; q, v) = -Hv$. An energy HVP and a force JVP therefore differ in sign.

Table 5: Derivative requests and contracts. Shapes use d flattened positions and p parameters.

Action	Definition	Shape	Required convention
Energy	$E_\theta(q)$	scalar	Total/per-atom energy and reduction
Force	$-\nabla_q E_\theta$	d	Position binding, atom order, sign
Energy HVP	$\nabla_q^2 E_\theta\, v$	d	Position and fixed input probe
Force JVP	$J_F v = -Hv$	d	Force convention and tangent
Force VJP	$J_F^\top w = -H^\top w$	d	Force cotangent and output axes
Parameter gradient	$\nabla_\theta E_\theta$	p	Parameter identity and energy cotangent
Force training	$-M^\top b_F$	p	$M = \partial g/\partial\theta$; loss cotangent
Cell derivative	Declared energy derivative under cell perturbation	task-specific	Fixed coordinates, volume, sign, axes

F.2 Force training and mixed derivatives

For a loss $\ell(E_\theta, F_\theta)$, let $a_E = \partial\ell/\partial E$, $b_F = \nabla_F \ell$, and $M = \partial g/\partial\theta \in \mathbb{R}^{d\times p}$. The chain rule gives

$$\nabla_\theta \ell = a_E \nabla_\theta E_\theta - M^\top b_F. \tag{8}$$

The cotangents are weights for these local derivative actions. Differentiating the weights again computes a different higher-order object. The position Hessian H and mixed Jacobian M have different differentiation variables and generally different shapes. A correct position HVP does not implement the parameter-backward action needed for force training.

For the exact scalar model $E_\theta(q) = \frac{1}{2}\theta q^2$, $F = -\theta q$, $Hv = \theta v$, and $M = q$. With $\ell = \frac{1}{2}(F - F^\star)^2$, $\partial_\theta \ell = -q(F - F^\star)$. At $q = 2$, $\theta = 3$, $F^\star = -5$, and $v = 1$, the force is -6, the HVP is 3, and the parameter gradient is 2. These values show why the HVP cannot substitute for the training gradient.

F.3 Reduction, cell, and compound-output conventions

If $E = \sum_i e_i$, replacing the sum by its mean scales forces and HVPs by $1/N$. For batches, the contract distinguishes scalar summed energy from a vector of structure energies and specifies the differentiation cotangent.

Cell derivatives additionally depend on what remains fixed. An illustrative path holds fractional coordinates s_i fixed, writes $x_i = Cs_i$, and perturbs $C(\eta) = (I + \eta)C$. Reconstructing a stress-like

tensor from $\partial E/\partial \eta$ requires a volume and sign convention. Holding Cartesian positions fixed defines a different path.

The `efsm` label names energy, force, stress, and magnetic-moment outputs; it does not define the magnetic-moment formula. The model/task interface must resolve that output. The derivative identities assume differentiable functions in a region with valid declared tensor identities; changing discrete neighbor topology is outside that local smoothness assumption.

G Exact worked example and structured repair

G.1 Descriptor, units, and reconstruction

For Figures 2 and 4, $E(x) = x^3$, $x = 2$, and $v = 3$. The task fixes

$$E = x^3, \quad g = 3x^2, \quad F = -g, \quad h = 6xv, \qquad (E, F, h) = (8, -12, 36). \tag{9}$$

A forward unit produces energy and saved values, a gradient unit consumes their declared dependencies, and a second-order unit implements the directional action. Reconstruction exposes the required signs, shapes, and probe binding.

Table 6: Exact scalar repair examples. Each row gives a distinct composition and its corresponding repair target.

Composition	(E, F, h)	Diagnosis	Repair target
Correct	$(8, -12, 36)$	Reference agreement	Retain bindings
Use $+g$ for force	$(8, +12, 36)$	Force sign	Restore $F = -g$
Bind probe $v' = 1$	$(8, -12, 12)$	Probe mismatch	Bind requested $v = 3$
Detached g treated as constant	$(8, -12, 0)$	Lost second-order path	Preserve gradient graph
Reuse H from $x' = 1$	$(8, -12, 18)$	Stale evaluation point	Recompute at $x = 2$

The detached row describes a faulty wrapper substituting zero for a disconnected derivative; an AD library may instead raise an error. Both outcomes fail this task. In the stale row, the Hessian value is locally correct at $x' = 1$ but wrong for the composed evaluation point.

G.2 Reference computation without an explicit Hessian

This explanatory reference uses reverse-over-reverse differentiation. It preserves the gradient graph, uses the same position tensor at both stages, and holds the probe fixed.

```
import torch

def reference_bundle(positions, probe):
    x = positions.detach().clone().requires_grad_(True)
    v = probe.detach()
    assert x.shape == v.shape
    energy = (x ** 3).sum()
    gradient, = torch.autograd.grad(energy, x, create_graph=True)
    hvp, = torch.autograd.grad((gradient * v).sum(), x)
    return {"energy": energy.detach(),
            "force": -gradient.detach(),
            "hvp": hvp.detach()}

x = torch.tensor([2.0], dtype=torch.float64)
v = torch.tensor([3.0], dtype=torch.float64)
# Exact bundle: energy=8, force=[-12], hvp=[36].
```

Detaching final comparison outputs is compatible with this example; detaching the intermediate gradient would lose its second-order path. The example computes a position HVP, not the parameter-backward action in Equation 8.

G.3 Multivariate axis and probe errors

Let $E(q_1, q_2) = q_1^2 q_2$, $q = (2, 3)$, and $v = (1, -1)$. Then

$$\nabla E = (2q_1 q_2, q_1^2), \quad H = \begin{bmatrix} 2q_2 & 2q_1 \\ 2q_1 & 0 \end{bmatrix}, \quad (E, F, h) = (12, (-12, -4), (2, 4)). \tag{10}$$

Swapping force coordinates yields $(-4, -12)$ despite correct local derivatives. Replacing the probe by $(1, 1)$ yields $h = (10, 4)$; reusing the Hessian from $q' = (2, 1)$ yields $h = (-2, 4)$ with the original probe. These cases distinguish output ordering, probe content, and evaluation-point provenance.

G.4 Feedback and revalidation

An illustrative report for the scalar force-sign error is:

```
{
  "pass": false,
  "unit_id": "force_reconstruction",
  "role": "force",
  "phase": "backward",
  "layer": "composed_output",
  "metric": "max_absolute_error",
  "value": 24,
  "candidate": [12],
  "reference": [-12],
  "diagnosis": "Check force sign in reconstruction."
}
```

The exact residual is $|12 - (-12)| = 24$. Feedback identifies a repair target but does not uniquely prove a cause. After changing the sign, the full composition must pass again, including its HVP output.

H IR bookkeeping and composition contracts

H.1 Unit fields and saved-value identity

Table 7 expands the IR in Section 3. The fields specify unit identity, dependencies, and reconstruction.

Table 7: Expanded IR fields for composition and attribution.

Field	Information	Workflow use
Unit identifier	Stable program-local reference	Attribute code and feedback
Phase and role	AD stage and scientific path	Distinguish force and HVP units
Typed bindings	Identity, dtype, device, rank, shape, layout	Match unit inputs/outputs
Saved-tensor reference	Producer, output slot, evaluation context	Resolve the intended intermediate
Dependency edges	Cross-unit and cross-phase reads	Order execution and identify consumers
Reconstruction rule	Axes, signs, reductions, probe binding	Assemble scientific outputs
Enabled hooks	Check, preconditions, dependencies	Apply compatible consistency checks
Allowed transforms	Permitted implementation changes	Constrain synthesis and refinement
Specialization key	Shape, dtype, device, neighbor/boundary metadata	Define specialized execution domain

Saved-value identity can be represented illustratively by $(u_{\text{producer}}, s_{\text{slot}}, c_{\text{eval}})$. The evaluation context distinguishes the inputs/model state that produced a value. Equal shapes and dtypes do not imply equal provenance; a consumer must use the declared producer reference.

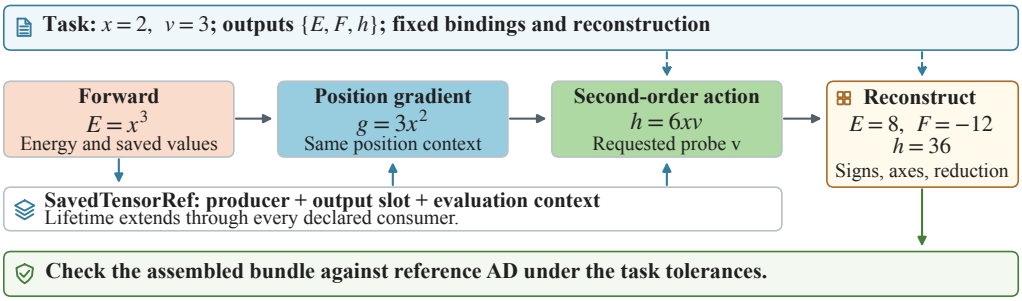


Figure 12: Illustrative dependency contract. Solid arrows carry values or dependencies; dashed arrows connect task declarations to reconstruction and checking. Saved-value identity and the requested probe remain visible across AD stages.

H.2 Scheduling and buffer lifetime

Execution respects the dependency graph, rather than phase labels alone. Saved storage remains live until its last declared consumer completes. Premature buffer reuse can corrupt backward or second-order outputs after the forward result has already been returned.

An upstream repair invalidates previously captured downstream values. The following pseudocode expresses the required fresh composition:

```
values = {}
for unit in topological_order(ir.dependency_edges):
    inputs = resolve_declared_bindings(unit, task, values)
    validate_types_shapes_and_provenance(unit, inputs)
    outputs = execute(unit.candidate, inputs)
    values.update(bind_declared_outputs(unit, outputs))
bundle = reconstruct(task.targets, ir.rules, values)
report = verify(bundle, reference_bundle, task)
```

Local oracle checks help isolate units but cannot replace the final reconstruction and task check.

H.3 Partition and specialization boundaries

Saved-tensor boundaries expose cross-phase dependencies; reduction boundaries expose aggregation domains; neighbor gather/scatter boundaries expose edge-to-atom mappings. These are semantic boundaries, not mandatory launch boundaries: permitted fusion can combine units while retaining their observable relations.

A neighbor-layout change can invalidate specialization even with unchanged atom count. Dtype and device changes likewise cannot inherit empirical acceptance automatically. Reuse follows the declared specialization domain and applicable checks.

I Agent interfaces and failure handling

I.1 Role inputs and outputs

The interfaces below expand Figure 3. The verifier is an execution gate; its numerical decision need not be delegated to an LLM.

Table 8: Role interfaces and their input/output contracts.

Role/component	Receives	Produces
Semantic lifting	Descriptor, trace, example inputs	Resolved bindings, IR, reconstruction
Synthesis worker	Unit, dependencies, local contract, feedback	Candidate implementation or unit failure
Composition	Candidate units, saved-value references, rules	Executable derivative artifact
Task verifier	Artifact, reference, inputs, tolerances, hooks	Pass/fail report with role, phase, metric
Repair	Rejected candidate, IR, failure report	Revised candidate for composed checking
Hardware refinement	Accepted artifact, device profile, allowed transforms	Proposal for the same task gate

An illustrative synthesis instruction asks a worker to implement its typed unit while preserving the declared role and saved dependencies, and to return its output mapping. Repair additionally names the failed scientific output and metric. Refinement starts from an accepted artifact and receives the target GPU and permitted transforms.

I.2 Rejection and termination

Ambiguous bindings, incompatible hook preconditions, or unstable reference outputs prevent a well-defined task comparison and require task rejection or resolution. Compilation/execution errors, output mismatches, non-finite values, and tolerance violations reject candidates. Looser tolerances do not resolve a misbound task or unstable reference.

Generation succeeds only after a composed pass; budget exhaustion without one is generation failure. Refinement retains a usable incumbent when a proposal fails or does not improve latency.

I.3 Attribution and feedback

A report can include failed output, residual, failing index, unit identifier, role, and AD phase. These narrow the repair target but do not uniquely identify defects: force mismatch may arise from sign, reduction, saved-value, or coordinate errors. A diagnostic hypothesis is checked through fresh composed execution.

J Hardware transformations and acceptance-preserving updates

J.1 Transformations and semantic obligations

Refinement changes an accepted artifact's implementation while preserving its task-facing meaning. Table 9 expands the transformation families in Section 3.

Table 9: Hardware transformation mechanisms and contract obligations.

Transform	Intended effect	Contract obligation
Tiling	Change work distribution and local reuse	Preserve coverage, tail masks, reductions
Memory layout	Change traversal, strides, storage	Preserve logical axes and saved-value meaning
Fusion	Reduce materialization and launch boundaries	Preserve dependencies and reconstruction
Vectorization	Process multiple elements per access/instruction	Respect alignment, valid lanes, dtype
Staging	Reuse values in faster local storage	Preserve synchronization and lifetimes
Launch configuration	Change blocks or warps	Preserve coverage and valid communication
Reduction organization	Change accumulation order/grouping	Retain reduction domain and pass tolerances

Larger tiles can increase register or shared-memory demand; fusion can extend saved-value lifetimes. Reduction-order changes can alter floating-point rounding without changing the real-valued formula. Each passing proposal is profiled before the replacement decision.

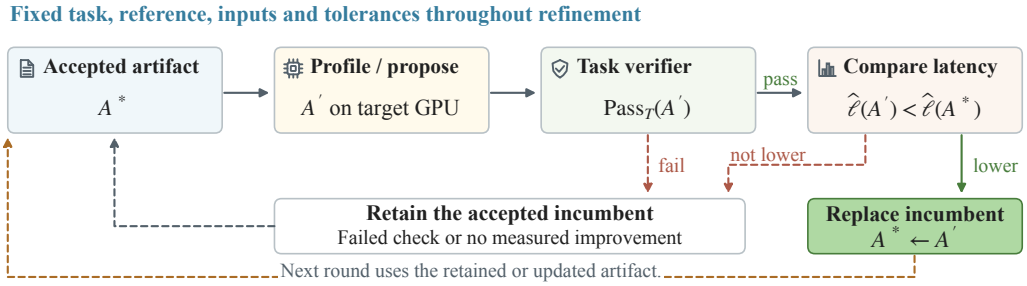


Figure 13: Refinement decisions from Algorithm 1. Only a passing proposal with lower measured latency replaces the incumbent.

J.2 Consequences of the replacement rule

For fixed task, reference, verification inputs, and tolerances, an initially passing incumbent remains empirically accepted under Equation 4: an iteration retains it or replaces it with another passing artifact. The two branches of the rule preserve this acceptance invariant.

The retained incumbent's recorded comparison latency is non-increasing under the same measurement protocol.

Table 10: Refinement outcome matrix. The incumbent is already accepted.

Proposal check	Latency	Decision
Fails task gate	Not needed for replacement	Retain incumbent
Passes task gate	Lower measured latency	Replace and update profile
Passes task gate	Equal or higher latency	Retain incumbent
Compilation/execution fails	Unavailable	Retain incumbent

J.3 Device and profiling interpretation

The same scientific contract can apply across GPUs, while the fastest configuration can differ. Per-device profiling relates traffic, occupancy, and launch overhead to the transform choices summarized in Table 9.

K Verification diagnostics and error taxonomy

K.1 Residuals and tolerances

For output y, let $a_j = |y_j^{\text{cand}} - y_j^{\text{ref}}|$ and $\tau_j = \texttt{atol}_y + \texttt{rtol}_y |y_j^{\text{ref}}|$. After shape, dtype, and finite-value checks,

$$\text{NumericPass}_y \iff \bigwedge_j (a_j \le \tau_j). \tag{11}$$

Useful summaries include maximum absolute residual and failing indices. If all $\tau_j > 0$, $\max_j (a_j/\tau_j) \le 1$ is equivalent to the predicate. Zero thresholds require exact agreement, without division. A small mean residual cannot hide a failing element.

The absolute term controls error near zero, while the relative term scales with reference magnitude. NaNs require explicit finite-value rejection.

Table 11: Qualitative failure taxonomy with diagnostic evidence and repair targets.

Failure	Consequence	Evidence	Repair target
Force sign	Wrong sign	Force residual	Reconstruction
Probe binding	Wrong directional action	HVP residual	Probe identity
Saved-value reuse	Wrong evaluation context	Residual and dependency metadata	Producer/lifetime
Axis permutation	Wrong output ordering	Elementwise mismatch	Axis mapping
Reduction mismatch	Wrong scale or domain	Aggregation residual	Reduction rule
Detached AD path	Missing second-order term	AD error or HVP residual	Gradient graph
Invalid alias	Overwritten consumer input	Residual and alias policy	Materialization
Tail/index error	Boundary-element mismatch	Failing indices	Masks/coverage
Non-finite value	Undefined output	Finite-value check	Numerical path

K.2 Consistency hooks and reference agreement

Translation checks require the corresponding invariance and boundary convention; HVP symmetry requires smooth scalar energy; rotation checks require declared output transformation laws. A hook outside its preconditions can reject a valid task.

Consistency alone is insufficient. Returning zero for every HVP satisfies $u^\top h(v) = v^\top h(u) = 0$ but fails a nonzero reference Hessian action. Reversing all force signs can preserve a zero total force. Reference comparisons and compatible physical hooks therefore have complementary diagnostic roles.

Metadata and local checks help locate the first divergence in a failing composition.

L Metric definitions and analytical cost accounting

L.1 Success, variability, and speedup

For seed s with n_s eligible attempts and k_s accepted artifacts, $p_s = 100k_s/n_s$. The unweighted mean and sample SD over S seeds are

$$\bar{p} = \frac{1}{S}\sum_s p_s, \qquad \mathrm{SD}(p) = \sqrt{\frac{1}{S-1}\sum_s (p_s - \bar{p})^2}. \tag{12}$$

This seed mean differs from pooled success $100\sum_s k_s / \sum_s n_s$ when denominators differ.

Speedup is $s = t_{\mathrm{ref}}/t_{\mathrm{cand}}$; latency reduction is $1 - t_{\mathrm{cand}}/t_{\mathrm{ref}}$. A twofold speedup thus corresponds to a 50% latency reduction. For the same workload and eager reference,

$$s_{\mathrm{VG/TC}} = \frac{t_{\mathrm{TC}}}{t_{\mathrm{VG}}} = \frac{s_{\mathrm{VG/eager}}}{s_{\mathrm{TC/eager}}}. \tag{13}$$

Appendix M derives these ratios from retained source coordinates before display rounding.

L.2 Sequential additions and unpaired extrema

The component means $78, 84, 86, 88$ percent yield sequential differences $6, 2, 2$ percentage points and a 10-point endpoint difference. These differences follow the cumulative component ordering. Pre/post suite minima may belong to different operators; their quotient is not a matched refinement effect. Paired comparison requires the same operator, input, and timing context.

L.3 Conditional and total costs

For successful attempts $\mathcal{S}$, failed attempts $\mathcal{F}$, and shared overhead,

$$C_{\mathrm{campaign}} = \sum_{i\in\mathcal{S}} C_i + \sum_{j\in\mathcal{F}} C_j + C_{\mathrm{shared}}. \tag{14}$$

A successful-case mean does not determine the second or third terms. GPU-hours require device time across workers, rather than elapsed time containing API waits. Token accounting requires the included exchanges to be specified.

Current MatRIS success rises from 72 to 88 percent, a 16-point difference. Its cost labels follow the existing means:

$$1 - \frac{2.48}{4.20} \approx 40.95\%, \quad 1 - \frac{312}{467} \approx 33.19\%, \quad 1 - \frac{48.6}{73.2} \approx 33.61\%. \tag{15}$$

These explain the displayed 41.0%, 33.2%, and 33.6% reductions.

L.4 Break-even interpretation

Let $\Delta t = t_{\mathrm{ref}} - t_A > 0$ and suppose the complete incremental offline time is known. Under the strict serialized break-even inequality in Appendix E, the smallest integer invocation count is

$$N_{\min} = \left\lfloor \frac{t_{\mathrm{offline}}}{\Delta t} \right\rfloor + 1. \tag{16}$$

If $\Delta t \leq 0$, repeated execution cannot amortize positive offline cost in this model. Variable shapes or overhead need richer accounting.

M Numerical values and figure reproduction

These tables collect the numerical values underlying the figures. Component values appear in Table 2; the tables below give derivative success, hardware speedup, operator latency, application performance, and generation cost. Analytical ratios are computed before display rounding.

M.1 Derivative-generation success

Rows group each method by forward, backward, and double-backward targets.

Table 12: Generation success (%) underlying Figure 1. Endpoint percentages by method and derivative target.

Method	Target	L1	L2	L3
AutoTriton	Forward	100	100	88
AutoTriton	Backward	78	69	54
AutoTriton	Double backward	72	64	49
KernelAgent	Forward	100	100	100
KernelAgent	Backward	98	94	92
KernelAgent	Double backward	95	84	78
Verigrad	Forward	100	100	100
Verigrad	Backward	100	100	98
Verigrad	Double backward	97	94	88

M.2 Cross-device speedup envelopes

Ranges summarize suite extrema relative to eager. Pre/post endpoints are unpaired.

Table 13: Suite speedup endpoints ($\times$ eager) underlying Figure 6. Suite minima and maxima before and after refinement, over three seeds.

GPU	Pre min	Pre max	Post min	Post max
RTX 3090	1.18	3.93	1.29	6.85
A100	1.07	3.68	1.21	6.45
A800	1.06	3.66	1.21	6.46
H100	1.21	4.07	1.32	6.56

M.3 Operator latency records

The eight records below match Figure 7. Recovered milliseconds are converted to microseconds. E/O and I/O denote eager/optimized and initial/optimized latency ratios within a row.

Table 14: RTX 3090 operator latencies (μs) and ratios from Figure 7.

Operator	Eager	Compiled	Initial	Optimized	E/O	I/O
Segment softmax	26.7	16.9	6.8	3.9	6.85	1.74
Fused SiLU	6.2	5.5	3.8	3.7	1.68	1.03
Edge fusion	5.5	4.3	3.7	3.5	1.57	1.06
Stress loss	9.4	7.1	6.2	6.0	1.57	1.03
Force loss	9.4	7.0	6.2	6.0	1.57	1.03
Energy-to-stress	5.9	4.2	3.6	3.3	1.79	1.09
Energy-to-force	4.5	4.2	3.8	3.5	1.29	1.09
Residual add	4.4	4.0	3.6	3.4	1.29	1.06

M.4 Twelve integrated MatRIS workloads

The table gives eager-relative speedups and derived VG/TC ratios in Figure 8. Ratios below one on energy-only inference for both models indicate faster compiled execution. Rows follow the displayed OAM/MP and `e`/`ef`/`efsm` order.

Table 15: Integrated speedups ($\times$) underlying Figure 8. TC and VG denote compiled and VERIGRAD execution.

Mode	Model	Bundle	TC/eager	VG/eager	VG/TC
Inference	OAM	`e`	1.66	1.42	0.85
Inference	OAM	`ef`	1.18	1.36	1.15
Inference	OAM	`efsm`	1.22	1.55	1.27
Inference	MP	`e`	1.77	1.51	0.85
Inference	MP	`ef`	1.27	1.55	1.22
Inference	MP	`efsm`	1.22	1.39	1.14
AdamW training	OAM	`e`	1.26	1.48	1.18
AdamW training	OAM	`ef`	1.17	1.33	1.14
AdamW training	OAM	`efsm`	1.24	1.47	1.19
AdamW training	MP	`e`	1.25	1.40	1.12
AdamW training	MP	`ef`	1.25	1.50	1.20
AdamW training	MP	`efsm`	1.24	1.47	1.19

M.5 Additional architecture-style configurations

Figure 9 reports latency and generation success separately for the two architecture-style configurations.

Table 16: Additional full-output inference records from Figure 9: ms latencies, % success, and latency ratios.

Configuration	Eager	TC	VG	Success	Eager/VG	TC/VG
MACE-style	142.6	118.4	98.7	78.1	1.44	1.20
NequIP/Allegro-style	128.9	109.2	89.4	82.3	1.44	1.22

M.6 Success and conditional generation cost

Success is 72 ± 3 and 88 ± 2 percent. Success/round dispersion is SD over three seeds; the cost metrics use the supplied dispersion described in Appendix A. Rounds, time, and tokens are per successful operator.

Table 17: MatRIS semantic-lifting aggregates underlying Figure 10. Changes are computed from the setting means.

Metric	Without lifting	With lifting	Change
Success (%)	72 ± 3	88 ± 2	$+16$ pp
Interaction rounds	4.20 ± 0.34	2.48 ± 0.15	-41.0%
Time (s)	467 ± 52	312 ± 28	-33.2%
Tokens (K)	73.2 ± 8.9	48.6 ± 5.2	-33.6%

M.7 Absolute application latencies

KernelAgent uses local-oracle checks and finite-output admission; VERIGRAD uses composed-task verification (Appendix D).

Table 18: Application latencies (ms) underlying Figure 11. Measurements use each method's admission rule.

Model	Bundle	Mode	Eager	Compiled	KernelAgent	Verigrad
OAM	`efsm`	Inference	113.03	92.40	85.10	72.92
OAM	`ef`	Training	328.42	281.60	263.80	246.93
MP	`efsm`	Inference	107.91	88.70	83.50	77.64
MP	`ef`	Training	327.74	263.10	242.70	218.49

M.8 Source provenance and reproduction of presentation assets

The tables use the same inputs as their figures: current derivative/component aggregates, current MatRIS generation-cost aggregates, and values recovered from archived vector plots. Architecture-style records use their existing renderer constants. Recoverable source precision is 0.0001 ms for operators and 0.01 ms for archived absolute application latencies. Ratios are computed before two-decimal display rounding.

The source package contains the manuscript, bibliography, official style, checklist, figure PDFs/SVGs, plotting scripts, and aggregate inputs. Tables are generated from these inputs. With plotting dependencies, fonts, and LaTeX available, the document assets can be rebuilt in this order:

```
python scripts/build_appendix_results.py
python scripts/redraw_current_figures.py
latexmk -pdf -interaction=nonstopmode -halt-on-error neurips_2026.tex
```

These scripts rebuild the figures, tables, and manuscript. Appendix A summarizes the experimental records and protocol fields.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction identify the derivative-task contract, composed verification, and hardware refinement. The evaluation covers KernelBench derivative generation and integrated MLIP workloads.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 6 discusses reference quality, verification-input coverage, hardware extensions, and resource analysis. Appendix A summarizes protocol records, and Appendix E develops the cost and amortization model.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper defines task acceptance and derives derivative identities and a replacement rule, but claims no new theorem or formal correctness proof.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [No]

   Justification: Section 4 reports the available configurations and aggregates. Appendix A identifies unavailable task IDs, per-run records, baseline revisions, and exact budget fields needed for full reproduction.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: No confirmed public code/data release or complete run-level reproduction package is provided. The available experimental protocol and its missing fields are stated in Appendix A.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [No]

   Justification: The paper gives the primary platform, timing repetitions, workload bundles, and optimizer type. Exact input structures, baseline revisions, and cross-family target mappings are unavailable (Appendix A).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: The component ablation and MatRIS success/round summaries report available three-seed standard deviations. Full agent per-cell variability and repeated-run latency dispersion are unavailable; no significance claim is made (Appendix A).

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [No]

   Justification: The evaluation identifies four GPU types and the primary RTX 3090 software stack. Complete device variants, per-stage/total GPU-hours, and cross-device software details are unavailable (Appendix A).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research concerns kernel generation for scientific derivatives and involves no human subjects or private participant data. Section 6 discusses numerical-error risks and compute costs.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Section 6 discusses reduced scientific-computing cost, errors in derivative outputs, and compute consumed by generation.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The work does not release high-risk pretrained models, image generators, scraped datasets, or human-subject data.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [No]

    Justification: The paper credits existing model families, compilers, and kernel agents through citations. A complete version-specific inventory of asset licenses and terms is not provided.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [No]

    Justification: The paper describes the task contract and workflow, but no confirmed public artifact release with complete code, generated kernels, and reproduction documentation is provided.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper involves no crowdsourcing, user studies, or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper involves no human subjects or participant data, so IRB review is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLM-based synthesis and refinement are core methods described in Section 3. Section 4 identifies the `GPT-5.4 xhigh` backend for VERIGRAD and the matched KernelAgent comparison.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.