Meaning on a Video Card: Scalable, Lossless Geometric Primitives

John Vaught
Synoros
ORCID: 0009-0006-0179-9522

Abstract

A GPU is engineered to rasterize triangles and decode video. Exact, lossless representation of non-Euclidean structure was never one of its design goals, and geometric machine learning re-implements the same delicate primitives project by project, re-encountering their numerical pitfalls each time. `holonomy_lib` is a PyTorch library of geometric and spectral primitives (Riemannian manifolds, hyperbolic graph operations, spectral graph theory, discrete Ricci curvature, computational topology) built to close that gap, a trusted substrate for learning geometric structure directly, made to hold under `float64`, automatic differentiation, and GPU execution.

Several of its components surfaced results worth recording. A multiplicative spectral-shift factor is silently dropped from the hyperbolic heat-kernel dimension recursion under automatic differentiation. A hand-derived closed form for the $\mathbb{H}^5$ heat kernel is faster and about three orders of magnitude more precise than that recursion. An arcsinh reparameterization of hyperbolic distance removes two distinct backward-`NaN` modes without an ε hyperparameter. A curvature-sign dispatch lets a continuous, learnable per-point curvature κ cross zero during training without re-instantiation, improving on discrete curvature gating in a controlled identifiability task. A reference-free dual-check method, computable from the object alone, certifies such primitives where no reference solution exists, and caught the dropped factor.

Two further capabilities instrument the geometry in use. The first is a spectral-dimension estimator that detects representational collapse, and the second is a content-addressable provenance layer that makes activation tracing and ablation native operations. Together they say that nonlinear, non-Euclidean embedding (learnable, per-point, and free to cross zero) is practically viable on commodity GPUs. Whether structure made directly available shortens the route language models take to the same geometry is the open wager the library is built to test.

Preprint.

1 Introduction

> *When I first heard Sutton say that large language models are bitter too, it crystallized the idea I had been circling in theory: that what a language model learns is the geometry of meaning (the shape of how things relate) and that it is made to learn this only indirectly, as an artifact of compressing how we communicate, rather than from the shape itself.*

Sutton [2019] argued that across six decades of artificial intelligence, general methods that scale with computation (search and learning) have consistently overtaken systems built from human-encoded domain knowledge. The knowledge-engineering approach wins in the short run and is overtaken in the long run. This is the bitter lesson. In a 2025 interview [Sutton and Patel, 2025], Sutton extended the argument to current large language models, which learn by imitating a finite store of human-generated text and do not learn continually from their own experience, so he expects systems that learn from experience and computation to eventually supersede them, another turn of the same lesson, with the models that depend most on human-produced data being the ones overtaken.

Language is itself a human artifact, an engineered, lossy intermediate representation of the world. Models trained on it do acquire non-linguistic structure, such as linear analogy geometry in word embeddings [Mikolov et al., 2013], the Fourier-basis representation a small transformer forms to do modular addition [Nanda et al., 2023], and the geometric addition circuit a production model uses in place of the textbook algorithm [Lindsey et al., 2025]. But that structure is acquired *indirectly*, reconstructed from the statistics of text, and at a cost in data and compute far above what the structure itself requires. If the bitter lesson applies one level down, the language channel is the human-encoded prior that scale will eventually route around.

The compute that powers that scale is no different. In practice scale means the GPU, and the GPU is a human-engineered artifact built to rasterize triangles and decode video. The general methods that overtake hand-built knowledge run on hardware hand-built for graphics, dense regular arithmetic at high throughput, where exact, lossless representation of structure was never a design goal. The substrate the bitter lesson runs on is itself a domain-specific human prior. Geometric machinery that means to ride the same hardware therefore carries a concrete obligation, to stay numerically exact while fitting what a video card does well.

This lab was founded on a wager that this concept can be acted on, that geometric and algorithmic structure can be made *directly available* to learning (as differentiable, composable primitives a model optimizes over and through). The working hypothesis of this substrate research program is that supplying correct geometric machinery, and letting scale and search act on it directly, reaches the structure that language models acquire slowly and as a side effect by a shorter and less wasteful route. The program is pursued in a sibling project (`synoros-substrate`) and shares the direction of an active literature on curvature-aware representation learning [Nickel and Kiela, 2017, Gu et al., 2019, Bachmann et al., 2020, Giovanni et al., 2022, Guo et al., 2025]. A learner that operates on a geometric substrate, however, inherits every error in that substrate. The substrate has to be correct first.

The actual reason for curved geometry in representation is target-fidelity [Vaught, 2026]. The familiar version (spacetime is hyperbolic, so embeddings should be hyperbolic) is not the true justification. Modeling meaning is, with one label peeled away, modeling the world that meaning describes, and a model of it must span the regimes that world presents, where a linear-algebra default supplies only the single flat one. Aircraft engineers studied birds without claiming aircraft are birds: they took the features that make flight work and built systems that respect them. The analogue here is a substrate that spans curvature (spherical, flat, hyperbolic, and the per-point mixtures and sign changes between them) because a faithful model of the target cannot be confined to one. The target is the structure of relations, learnable from text, images, or concepts alike, since what is fit is the shape of those relations, independent of the semantics that express them.

`holonomy_lib` is that substrate layer. It consolidates the mathematics that geometric machine learning otherwise re-implements project by project. It spans Riemannian manifolds (fixed-rank, SPD, the hyperboloid model, the κ-stereographic model, the Lorentzian $(1, n-1)$ model, product manifolds, and a per-point-κ heterogeneous manifold); hyperbolic graph operations (the Fréchet mean [Karcher, 1977], manifold-aware inner products [Pennec, 2006], hyperbolic Laplacian eigenmaps, the $\mathbb{H}^n$ heat kernel); spectral graph theory (combinatorial, normalized, signed and magnetic

Laplacians, diffusion maps, effective resistance, spectral dimension); discrete Ricci curvature and flow; tensor decompositions; Riemannian optimization [Absil et al., 2008]; simplicial topology and batched persistent homology; cellular sheaves; SO(3) primitives; and a content-addressable provenance layer. Everything is GPU-native and batched-first; every numerical constant is either derived from inputs, marked as a universal invariant, or cataloged with a documented procedure and scale of validity; every public primitive carries a citation to the source that defines its mathematics; and a static audit enforces the constant discipline in continuous integration.

Optimizing that substrate took work in seven specific places, each between mathematical correctness and implementation correctness under `float64`, automatic differentiation, and GPU execution. Nonlinear, non-Euclidean embedding (learnable curvature that is per-point and free to cross zero) stays numerically exact and runs at practical speed on commodity hardware (an AMD RX 9060 XT under ROCm, an NVIDIA RTX 3060 under CUDA; no datacenter accelerator), which is what makes it viable in practice. A substrate earns the trust to build on only once these gaps close, and the sections that follow close them, covering correctness (Section 3), precision and speed (Section 4), new primitives (Section 5), and a provenance layer that keeps the substrate auditable (Section 6). Section 7 returns to the framing.

Artifacts. Every result below is backed by a symbolic verification script, a numerical demonstration, and a consolidated write-up in the `holonomy_lib` repository under `notes/verification/`, `notes/strengthening/`, and `notes/validation/`. Specific paths appear inline.

2 Background

The κ-stereographic model. A single model of constant sectional curvature $\kappa \in \mathbb{R}$ interpolates the sphere ($\kappa > 0$), Euclidean space ($\kappa = 0$), and hyperbolic space ($\kappa < 0$). Distances and the exponential/logarithm maps are expressed through the curvature-dependent trigonometric functions $\tan_\kappa$ and its inverse $\tan_\kappa^{-1}$, built on Möbius gyrovector operations [Ungar, 2008]. Bachmann et al. [2020] introduced this model to graph convolutional networks with κ a single learnable scalar per layer, recovering the Euclidean network as $\kappa \to 0$. Mixed-curvature representations take products of such spaces with fixed per-factor curvatures [Gu et al., 2019, Skopek et al., 2020].

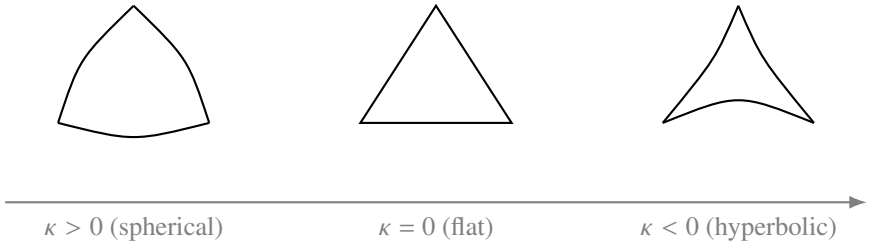

Figure 1: One κ-stereographic model spans the curvature spectrum: a geodesic triangle has angle sum $> \pi$ on the sphere ($\kappa > 0$), $= \pi$ in the flat limit, and $< \pi$ in hyperbolic space ($\kappa < 0$). The per-point-κ manifold (Section 5.1) lets κ vary across points and cross zero during training (Section 4.3) within one model.

The hyperbolic heat kernel. The heat kernel on $\mathbb{H}^n$ is a radial function $k_t(r)$ solving the heat equation $\partial_t k = \Delta k$ with a point-mass initial condition [Davies, 1989]. Davies and Mandouvalos [1988] give the $\mathbb{H}^3$ closed form. Across dimension, the kernel obeys a recursion that raises n by two by applying the operator $\frac{1}{\sinh r}\,\partial_r$ [Grigor'yan and Noguchi, 1998, Grigor'yan, 2009, Yu and Zhao, 2018]. Both appear in Sections 3.1 and 4.1.

Per-point curvature. Beyond a single learnable κ, several constructions allow curvature to vary, including products with spherically symmetric factors [Giovanni et al., 2022], reinforcement-learning control of a global κ [Fu et al., 2021], and, closest to the per-node setting, a mixture of Riemannian experts that gates each node over a *discrete* bank of constant-curvature spaces [Guo et al., 2025]. Section 5.1 contrasts a continuous per-point κ against this discrete gating.

Numerical setting. All primitives are implemented on `torch.Tensor` with automatic differentiation [Paszke et al., 2019]. Where an established library exists, the implementation is checked against it, using `geoopt` for Riemannian operations [Kochurov et al., 2020], whose ε-clamp idiom appears in Section 3.2, and the `safe_*` helper pattern shared with PyTorch3D [Ravi et al., 2020].

3 Implementation correctness

Each result here is a place where a formula correct on paper misbehaves once it runs under `float64` and automatic differentiation. One is a multiplicative factor that autograd cannot see, dropped from the heat-kernel dimension recursion (Section 3.1). Another is a hyperbolic distance whose textbook form carries a boundary derivative singularity and catastrophic cancellation, both removed by an algebraically equivalent reparameterization (Section 3.2). A third is a reference-free correctness gate that certifies such primitives where no reference solution exists, and that caught the first (Section 3.3).

3.1 An autograd pitfall in the heat-kernel dimension recursion

The recursion that raises the hyperbolic heat kernel by two dimensions, stated correctly in the literature [Grigor'yan and Noguchi, 1998, Grigor'yan, 2009, Yu and Zhao, 2018], is

$$k^{n+2}(t,r) \;=\; -\frac{e^{-nt}}{2\pi \sinh r}\,\partial_r\, k^{n}(t,r). \tag{1}$$

The factor e^{-nt} is a spectral shift, multiplicative and *constant in* r. This is exactly what makes it easy to drop when porting (1) to automatic differentiation. The natural implementation,

```
grad = torch.autograd.grad(k_n(t, r).sum(), r)[0];  return -grad /
                    (2*math.pi*torch.sinh(r)),
```

computes $\partial_r k^{n}$ faithfully and then divides by $2\pi \sinh r$, but never multiplies by e^{-nt}, because that factor does not depend on the differentiated variable r and so is invisible to `torch.autograd.grad`. The factor has to be applied by hand, outside the autograd call. The corrected kernel multiplies by $e^{-n_{\text{prev}}t}$ explicitly and clamps $\sinh r$ at `finfo.tiny` for the $r \to 0$ limit.

The error has a clean signature. At a sample $(t,r) = (0.5, 1.0)$ the naive recursion returns exactly $e^{3\cdot 0.5} = e^{1.5} \approx 4.4817$ times the correct value, the missing e^{nt}. Symbolic verification (`notes/verification/heat_kernel_recursion_sympy.py`) confirms this algebraically, where `sympy.simplify` reduces the heat-equation residual of the corrected k^5 to exact zero, while the naive k^5 leaves a nonzero residual.

When porting an analytical formula to autograd code, any multiplicative factor that depends on a *non-differentiated* variable (here t and the dimension n) must be applied by hand. Automatic differentiation reports only the derivative it computes, so a factor outside that derivative stays silently absent.

3.2 Equivalent mathematics, very different float behavior: arcsinh vs. arccosh

The textbook curvature-κ hyperbolic distance $d_\kappa(x,y) = \frac{1}{\sqrt{|\kappa|}} \operatorname{arccosh}\big(\kappa \langle x, y\rangle_{\mathcal{M}}\big)$ is exact on paper and unreliable in `float64` under autograd, for two distinct reasons:

1. *Boundary derivative singularity.* $\frac{d}{dz}\operatorname{arccosh}(z) = 1/\sqrt{z^2-1}$ diverges at $z = 1$, i.e. at the on-manifold self-pair $x = y$. The forward value $\operatorname{arccosh}(1) = 0$ is finite, but the backward pass propagates $\infty \cdot \partial z/\partial x = \texttt{NaN}$.
2. *Catastrophic cancellation.* For $x \approx y$, $z = \kappa\langle x, y\rangle_{\mathcal{M}} = \cosh(\alpha) = 1 + \alpha^2/2 + O(\alpha^4)$, so the information about a small distance α sits in the last bits of a number near 1, and $\operatorname{arccosh}(1+\delta) \approx \sqrt{2\delta}$ then amplifies the lost precision.

The equivalent reparameterization

$$d_\kappa(x,y) \;=\; \frac{2}{\sqrt{|\kappa|}}\,\operatorname{arcsinh}\!\left(\tfrac{1}{2}\sqrt{|\kappa|}\;\|y - x\|_{\mathcal{M}}\right) \tag{2}$$

Table 1: Recovered distance at small geodesic separation α, three implementations. The textbook form loses all bits below $\alpha \sim 10^{-8}$. The ε-clamp (`geoopt`-style, $\varepsilon = 10^{-5}$) returns the constant $\operatorname{arccosh}(1+\varepsilon) \approx \sqrt{2\varepsilon} \approx 4.47 \times 10^{-3}$ for *every* α below the clamp. Eq. (2) tracks the true value to floating-point precision. Source: `notes/strengthening/C3_arcsinh_worked_example_results.md`.

α	textbook arccosh	ε-clamp	arcsinh (Eq. 2)
10^{-4}	1.00×10^{-4}	4.47×10^{-3}	1.00×10^{-4}
10^{-6}	1.00×10^{-6}	4.47×10^{-3}	1.00×10^{-6}
10^{-8}	0 (all bits lost)	4.47×10^{-3}	1.00×10^{-8}
10^{-10}	0 (all bits lost)	4.47×10^{-3}	1.00×10^{-10}
0 ($x = y$)	0 fwd, `NaN` bwd	4.47×10^{-3}	0 fwd, 0 bwd

removes both. It computes $\|y-x\|^2_{\mathcal{M}}$ from coordinate differences (no subtraction near 1), and arcsinh is entire, with derivative $1/\sqrt{1+\arg^2}$ equal to 1 at the origin, with no boundary singularity. The half-angle identity $\cosh A - 1 = 2\sinh^2(A/2)$ makes (2) algebraically identical to the textbook form (`notes/verification/arcsinh_reparam_sympy.py`). Table 1 shows the practical difference in `float64`.

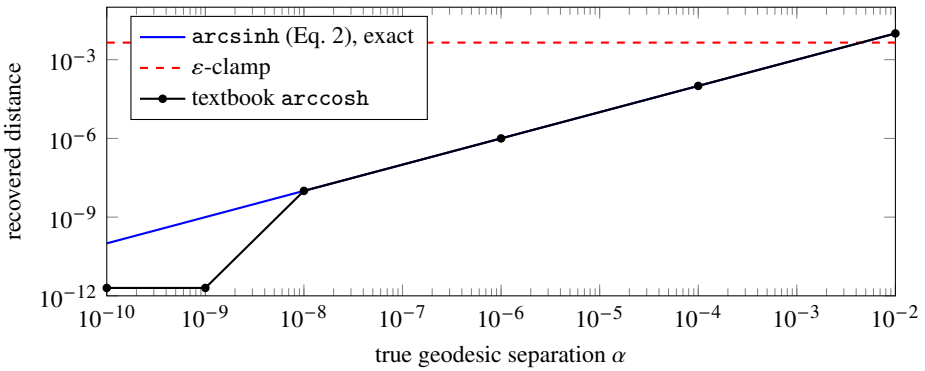

Figure 2: The precision cliff of Table 1 on log–log axes. `arcsinh` tracks the true separation to machine precision; the ε-clamp pins every small distance at a constant $\sim 4.5\times10^{-3}$ bias; the textbook `arccosh` form matches until $\alpha \sim 10^{-8}$, then loses all bits.

The ε-clamp removes the `NaN` at the price of a constant forward bias of $\sim 4.5 \times 10^{-3}$ and an extra hyperparameter. Eq. (2) removes the `NaN` and keeps the value exact, with one `safe_sqrt` (a `torch.where` guarding the $\sqrt{0}$ derivative) as its only cost, about 30% per-call overhead in isolation (batch 10^4, $n = 8$), shrinking to a small fraction of any realistic training step. The same reparameterization drives `log`. The `safe_*` idiom is folklore in `geoopt` and PyTorch3D [Kochurov et al., 2020, Ravi et al., 2020]. Applied consistently across the manifold API it keeps every boundary autograd-safe, and it reframes the choice between the two distance forms as a precision decision.

3.3 A reference-free correctness gate for operator-semigroup primitives

The bug in Section 3.1 was found by two checks that any correct operator semigroup S_t satisfies internally, each computable from the object alone, the only option for the higher-dimensional kernels, where no reference solution existed:

Operator-equation residual. S_t solves an evolution equation $\partial_t S = \mathcal{G}S$. Evaluate $\partial_t S - \mathcal{G}S$ by finite differences and report it relative to $\|\mathcal{G}S\|$. A correct implementation sits at the finite-difference truncation floor.

Conserved-quantity check. S_t conserves a known quantity Q (probability mass, row sum, trace, energy). Evaluate $Q(S_t) - Q$. A correct implementation sits at machine precision.

Table 2: Three corruptions of the graph heat kernel against the two checks. Each of the last two corruptions is invisible to one check and caught by the other. The two together form the gate.

Corruption	Operator residual	Mass conservation
spectral-shift $K \to e^{\alpha t} K$	fires (grows with t)	fires ($= e^{\alpha t} - 1$)
constant rescale $K \to c\,K$	**blind** (floor $\forall c$)	fires ($= \lvert c - 1 \rvert$)
mass-preserving $K \to K + \varepsilon L'$	fires ($\propto \varepsilon$)	**blind** (machine $\forall \varepsilon$)

Both are reference-free, which is what makes them usable for a *new* primitive. Applying them to a structurally different object, the graph heat kernel $K_t = e^{-tL}$ on a 24-node graph, where the PDE becomes the ODE $\dot{K} = -LK$ and probability mass becomes row mass $K_t\mathbf{1} = \mathbf{1}$ (`notes/validation/graph_heat_kernel_validation.py`), shows the two checks cover distinct failure modes (Table 2).

A constant rescale is in the null space of the residual (a scalar multiple of a solution to the homogeneous equation still solves it), but mass conservation catches it immediately. A mass-preserving perturbation by another Laplacian L' (so $L'\mathbf{1} = 0$) leaves every row sum unchanged but no longer solves the heat ODE, so the residual fires. The spectral-shift bug of Section 3.1 happens to trip both, which is why one check sufficed to *find* it. The table shows that in general both are needed. A closed-form spot check is a weaker third gate, constraining only the points evaluated, and serves as corroboration.

4 Precision and speed

4.1 A closed form for the $\mathbb{H}^5$ heat kernel

The operator-chain form for odd $n = 2m + 1$ [Grigor'yan, 2009, Thm. 8.21] is $p^{2m+1}(t, r) = \frac{(-1)^m}{(2\pi)^m}(4\pi t)^{-1/2}\big(\frac{1}{\sinh r}\partial_r\big)^m e^{-m^2 t - r^2/4t}$. Carrying out the two operator applications for $m = 2$ by hand (`notes/verification/heat_kernel_n5_sympy.py`) gives the closed form

$$k_t^5(r) = (4\pi t)^{-5/2}\, e^{-4t - r^2/4t}\, \frac{r^2 \sinh r + 2t\,(r\cosh r - \sinh r)}{\sinh^3 r}, \tag{3}$$

with the $r \to 0$ limit $k_t^5(0) = (4\pi t)^{-5/2} e^{-4t}\,(1 + 2t/3)$. Symbolic verification confirms that (3) equals one corrected application of (1) to k^3, satisfies the $\mathbb{H}^5$ heat equation, and matches the limit, all with residual exactly zero.

Using (3) directly, rather than two autograd-based recursion steps from $n = 3$, is both faster and more precise. Each `torch.autograd.grad` call accumulates floating-point noise, while the closed form is one straight evaluation. The PDE residual drops by about three orders of magnitude (10^{-8}–10^{-6} versus 10^{-5}–10^{-3}). Table 3 reports latency. The $n = 5$ closed form costs only slightly more than $n = 3$, an order of magnitude below the even-n quadrature-plus-recursion paths.

4.2 The same construction at $\mathbb{H}^7$, and an independent check

One further operator step gives the $\mathbb{H}^7$ closed form,

$$\begin{aligned} k_t^7(r) &= (4\pi t)^{-7/2}\, e^{-9t - r^2/4t}\, \frac{P(t,r)}{\sinh^5 r}, \\ P(t,r) &= r^3 \sinh^2 r + 6r^2 t\, \sinh r \cosh r \\ &\quad + (8t^2 - 6t)\, r \sinh^2 r + 12t^2 (r - \sinh r \cosh r). \end{aligned} \tag{4}$$

with $r \to 0$ limit $(4\pi t)^{-7/2} e^{-9t}(1 + 2t + 16t^2/15)$. It equals one corrected application of (1) to k^5 and the operator chain at $m = 3$, verified symbolically and numerically to $\sim 10^{-13}$, with an $\mathbb{H}^7$ heat-equation residual $\sim 10^{-12}$ (`notes/verification/heat_kernel_n7_sympy.py`). Used directly it costs essentially the same as the $n = 5$ closed form (Table 3), and it seeds the odd-n recursion at $n \geq 9$, where one recursion step adds ~ 0.8 ms forward+backward (compare $n = 9$ at 1.35 ms to the

Table 3: Heat-kernel latency, CPU `float64`, batch 4096. Commodity-GPU scaling is in Table 4 (Section 4.4). Source: `notes/strengthening/C1_C2_heat_kernel_bench_results.md`.

n	path	forward (ms)	fwd+bwd (ms)
3	Davies–Mandouvalos closed form	0.11	0.32
5	**hand-derived closed form (Eq. 3)**	**0.17**	**0.48**
7	**hand-derived closed form (Eq. 4)**	**0.19**	**0.53**
9	recursion from $n = 7$ (1 step)	0.59	1.35
6	recursion from $n = 2$ (2 steps)	11.2	21.7

$n = 7$ closed form at 0.53 ms), so the step's cost and its float-noise compounding are both avoided at $n = 7$.

A Crank–Nicolson solver for the radial heat equation $\partial_t u = \partial_r^2 u + (n-1)\coth(r)\,\partial_r u$, sharing none of the operator-chain machinery, gives an independent confirmation. Evolving each library kernel forward in time reproduces it to the scheme's truncation floor, 7.8×10^{-5} at $n = 5$, 1.8×10^{-4} at $n = 7$, 3.9×10^{-4} at $n = 9$ (`notes/validation/heat_kernel_crank_nicolson_results.md`). A wrong coefficient, the kind of error the spectral shift of Section 3.1 introduced, would show an $O(1)$ mismatch rather than 10^{-4}.

4.3 Letting a learnable curvature cross zero without retraining

When κ is a learnable parameter on the κ-stereographic model, the geometry (spherical / hyperbolic / Euclidean) can change during training. The map

$$f(\kappa, \alpha) = \frac{\tan_\kappa^{-1}(\sqrt{|\kappa|}\,\alpha)}{\sqrt{|\kappa|}} = \begin{cases} \arctan(\sqrt{\kappa}\,\alpha)/\sqrt{\kappa}, & \kappa > 0, \\ \alpha, & \kappa = 0, \\ \operatorname{arctanh}(\sqrt{|\kappa|}\,\alpha)/\sqrt{|\kappa|}, & \kappa < 0, \end{cases} \tag{5}$$

is a single analytic function of $\kappa \in \mathbb{R}$, with Taylor series $f(\kappa, \alpha) = \alpha \sum_{m \geq 0} (-\kappa\alpha^2)^m/(2m+1) = \alpha\,(1 - \kappa\alpha^2/3 + \kappa^2\alpha^4/5 - \cdots)$ converging for $|\kappa|\alpha^2 < 1$, exactly the manifold's distance-formula domain. The two closed forms express one analytic function, one for each side of zero. Symbolic verification (`notes/verification/kappa_crossing_sympy.py`) confirms that the two branches' values, first and second κ-derivatives all agree at $\kappa = 0$, and that an integral representation $f(\kappa, \alpha)/\alpha = \int_0^1 (1 + \kappa(t\alpha)^2)^{-1}\,dt$ bridges both signs without a conditional.

Realizing this through `torch.autograd` needs three pieces, since autograd requires two finite expressions to mask between rather than one analytic function. (1) Clamp $|\kappa|$ at `finfo.tiny` before the square root, so $\sqrt{|\kappa|}$ never vanishes and the scaled argument is tiny-but-nonzero; (2) `safe` versions of $\arctan(t)/t$ and $\operatorname{arctanh}(t)/t$ that short-circuit $t \leq 0$ to the analytic limit 1; (3) an outer `torch.where(κ > 0, ...)` selecting per element in forward and masking the gradient in backward. Inside the manifold's distance-formula domain, both branches are finite, so the dispatch is autograd-safe. Outside it (an input the manifold's own domain restriction excludes) the masked-out `arctanh` branch can leak a `NaN` through the backward.

A 100-step stochastic-gradient trajectory whose target oscillates between +0.7 and −0.7 (four sign crossings) keeps κ, its gradient, the distances, and the points all finite at every step, with no special-casing of $\kappa = 0$ (`notes/strengthening/C4_kappa_crossing_stress_results.md`). The alternatives are worse. A static-branch manifold that caches the sign at construction computes arctanh where it should compute arctan after a flip, a 1.38% relative error (0.4086 vs. the correct 0.4030); and a truncated unified Taylor series, the natural `where`-free alternative, still leaves $\sim 10^{-3}$ error with 32 terms near the domain boundary ($\sqrt{|\kappa|}\alpha = 0.95$) where the dispatch is exact. The dispatch evaluates both branches, but each is a single elementwise kernel and the Möbius machinery dominates and is branch-independent, so the measured overhead is well below the worst-case 2×, CPU ∼1.2–1.4× (batch 16384, $n = 8$, `float64`), and on a consumer AMD Radeon RX 9060 XT (gfx1200, ROCm 6.4, batch 65536, $n = 8$), 1.27× in `float64` and 1.58× in `float32`. Curvature-as-learnable-scalar is established [Bachmann et al., 2020, Skopek et al., 2020]. Those implementations leave the behavior

Table 4: Heat-kernel forward+backward latency (ms), `float64`, same-machine CPU vs. CUDA on an RTX 3060. GPU latency is near-flat in batch, CPU linear. The CPU/GPU ratio at 10^6 is ~ 80–110×. Source: `notes/strengthening/gpu_scaling_results.md`.

n	path	CPU 4×10^3	CPU 10^6	GPU 4×10^3	GPU 10^6
5	closed form	10.0	646	1.8	5.8
7	closed form	19.5	627	6.8	8.0
9	recursion (from n=7)	47	2320	9.7	24

at $\kappa = 0$ implicit, sidestepping it by initializing on one side and regularizing away from the boundary. The dispatch here crosses it, with the analyticity verified symbolically and the crossing exercised in training.

4.4 Scaling on a commodity GPU

The same kernels run at practical speed on consumer hardware. On an RTX 3060 (12 GB, `float64`, CUDA), forward+backward latency is near-flat in batch while the same-machine CPU scales linearly. From a batch of 4096 to ~ 10^6 (a 256× increase) the GPU time barely moves while the CPU time grows roughly proportionally (Table 4). A batch of a million hyperbolic heat-kernel evaluations costs on the GPU about what 4096 does (launch-bound, not compute-bound, well past 10^6 points), and the CPU/GPU ratio there reaches ~ 80–110×. With the κ-crossing dispatch measured on an AMD RX 9060 XT under ROCm (Section 4.3), the substrate's geometry runs on commodity cards of either vendor, no datacenter accelerator required. Consumer GPUs of this class rent by the hour, so a single researcher can run non-Euclidean embedding at 10^6-point scale without institutional compute.

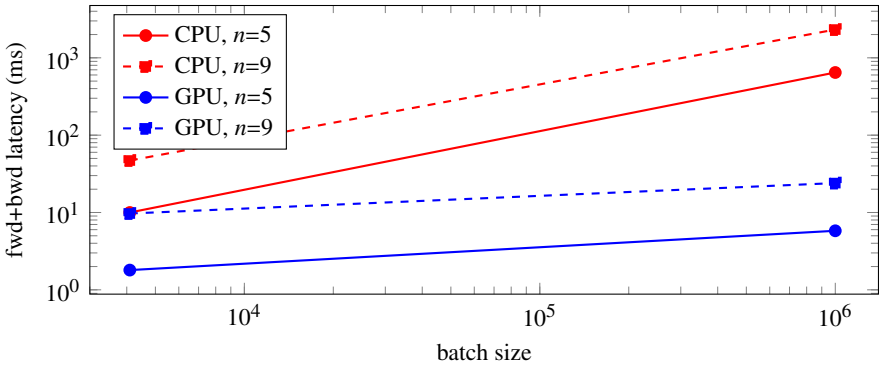

Figure 3: Commodity-GPU scaling (Table 4): from 4096 to 10^6 points the same-machine CPU latency grows roughly with the batch while the RTX 3060 stays near-flat, an ~ 80–110× gap at 10^6. Two measured batch sizes per curve.

5 New primitives

5.1 Continuous per-point curvature

`HeterogeneousKappaManifold` exposes each point's curvature as a continuous real-valued tensor. Autograd flows through κ directly, any $\kappa \in \mathbb{R}$ is representable, and there is no expert bank to choose. The closest prior art is discrete gating, in which each node mixes over a fixed bank $\{\kappa_1, \ldots, \kappa_K\}$ of constant-curvature spaces [Guo et al., 2025]. Its recovered curvature is confined to the convex hull of the bank, so the bank must be guessed before training. The class supplies the math (per-point κ-trig, exponential and logarithm maps, and pair distance) and leaves the κ parameterization to the caller. The κ-field-plus-per-point-residual parameterization used downstream belongs to the substrate project. The library provides the geometry beneath it.

Table 5: Identifiability task. Final fit loss and Pearson correlation between recovered and true per-node κ. Continuous per-point κ with the arithmetic-mean combiner attains the lowest loss and the best κ-recovery. The adversarial-bank discrete model pays a bank-choice penalty of $\sim 25\%$ in loss.

Variant	combiner / bank	final loss	κ-recovery corr.
single κ baseline	none	0.0098 ± 0.0046	N/A
discrete gating	friendly $\{-1, 0, +0.5\}$	0.0099 ± 0.0042	$+0.630 \pm 0.071$
discrete gating	adversarial $\{-2, +1.5\}$	0.0124 ± 0.0045	$+0.526 \pm 0.071$
continuous per-point κ	**arithmetic_mean**	$\mathbf{0.0065 \pm 0.0061}$	$\mathbf{+0.667 \pm 0.077}$
continuous per-point κ	harmonic_mean	0.0092 ± 0.0065	-0.001 ± 0.151
continuous per-point κ	max_magnitude	0.0069 ± 0.0055	$+0.250 \pm 0.198$

Table 6: Combiner characterization. A combiner that is smooth and well-defined at $\kappa = 0$ trains reliably. A non-smooth one trains poorly.

combiner	behavior at $\kappa = 0$	smoothness	fit on truth
`arithmetic_mean` (default)	Euclidean, well-defined	C^∞	best (matches truth)
`harmonic_mean`	needs `finfo.tiny` clamp	C^∞ away from 0	collapses (clamp)
`signed_geometric_mean`	sign jump at sum = 0	non-smooth	brittle across seeds
`max_magnitude`	jump at $\lvert\kappa_x\rvert = \lvert\kappa_y\rvert$	non-smooth	mediocre

A controlled identifiability task makes the comparison concrete (`notes/strengthening/C5_C6_synthetic_task.py`). It uses 90 nodes in three regions with true curvatures $\{-1.0, 0.0, +0.5\}$, ground-truth coordinates sampled per region, and pairwise distances under the true κ-field as embedding targets. Variants recover (coords, κ-field) from distances (Table 5, mean $\pm$ std over three seeds, 1500 epochs, dimension 3, on an AMD ROCm GPU).

Continuous-with-arithmetic-mean wins on both metrics in the truth-aligned case (the targets were generated under that combiner). The single-κ baseline ties the friendly-bank model on loss. With one degree of freedom it settles on a global $\kappa \approx -0.84$ and represents no per-node variation. Two caveats bound the result. The κ-recovery correlation tops out near 0.67 because at dimension 3 with 90 nodes several (coords, κ) configurations fit the distances about equally well. The ranking across variants holds, while the absolute number reflects the task's identifiability limit rather than the primitive's capacity. And training continuous κ takes care. A soft coordinate barrier ($\|x\| < 0.85$), a hard κ clamp ($|\kappa| < 1.5$), gradient clipping, and cosine learning-rate decay keep momentum from driving points outside the domain into a forward-`NaN`. Autograd-safety governs the forward and backward passes. Optimization stability is a separate requirement, met here by those four guards.

5.2 The pair-κ combiner abstraction

Computing a pairwise distance between two points of different curvature requires combining κ_x and κ_y into one effective pair-curvature, and that rule remains a free choice in the literature. The library exposes it as a callable $(\kappa_x, \kappa_y) \mapsto \kappa_{\text{eff}}$ with two built-in defaults and arbitrary override. Four combiners on the task above (Table 6) show the design axis. The empirical winners are smooth and well-defined at $\kappa = 0$. `arithmetic_mean` is both the well-motivated default and the best performer; `harmonic_mean` collapses when the truth includes $\kappa = 0$ because of the `finfo.tiny` guard; the non-smooth combiners (`signed_geometric_mean`'s sign, `max_magnitude`'s $|\cdot|$) are brittle under Adam, and a `safe_sqrt` fixes the boundary derivative but not an upstream non-smooth sign. The combiner is a callable the user sets, with `arithmetic_mean` as a smooth, well-motivated default.

5.3 Mixed-curvature product manifolds

For completeness the library includes `ProductManifold`, the well-established construction of a product of constant-curvature factors with a per-factor metric [Gu et al., 2019, Skopek et al., 2020]. The projection and retraction compose factor-wise over the existing single-manifold API. It reuses established geometry so the substrate covers the mixed-curvature setting.

Table 7: `spectral_dimension` against structures with known d_s, one window rule for all. The Sierpinski gasket is the canonical non-integer case [Rammal and Toulouse, 1983]. The lattice residuals are the finite-t approach of the return-probability slope to its asymptote.

structure	known d_s	recovered
1-D ring (cycle)	1.000	1.000
2-D torus	2.000	2.031
3-D torus	3.000	3.094
Sierpinski gasket	1.365	1.320

5.4 Spectral dimension as a collapse diagnostic

A representation that nominally occupies $\mathbb{R}^D$ can collapse onto a far lower-dimensional set. Detecting that is one of the questions the substrate program raises directly. The spectral dimension reads it off a Laplacian spectrum. With return probability $p(t) = \frac{1}{n}\sum_i e^{-t\lambda_i} \sim t^{-d_s/2}$, the estimator $d_s = -2\,\mathrm{d}\log p/\mathrm{d}\log t$ (`spectral.spectral_dimension`, after Rammal and Toulouse [1983]) recovers a possibly non-integer dimension. Under one window rule (the asymptotic tail of the return probability) it recovers integer lattice dimensions and the Sierpinski gasket's fractional $2\ln 3/\ln 5 \approx 1.365$ (Table 7, `notes/validation/spectral_dimension_results.md`). A point set collapsed to an effective line registers $d_s \approx 1$ whatever D is, the reading the substrate's observed drift toward low effective dimension calls for.

6 Content-addressable provenance for mechanistic interpretability

Studying whether and how geometric structure forms requires tracing which primitive produced which representation, and intervening on it. The library makes that native rather than an external instrumentation pass. Every primitive decorated with `@with_provenance` emits, inside a `record()` context, a node in a content-addressable Merkle DAG keyed by

$$\mathrm{hex}(\mathrm{call}) = \mathrm{sha256}\big(\mathrm{op\_id} + \mathrm{version} + \mathrm{canonical}(\mathrm{params}) + [\,\mathrm{hex}(\mathrm{input})\,]\big),$$

leaf tensors hashed by content (bytes, shape, dtype). The same operation on the same inputs yields the same hex, so identical sub-computations across runs share a node. The DAG exports to NetworkX for graph analysis, to a Pandas table of (op_id, params, hex, shape) records for SAELens-style feature training, and to JSON. Substitution at a hex (`registry.substitute`) applies activation patching, the TransformerLens intervention, to math primitives rather than network layers.

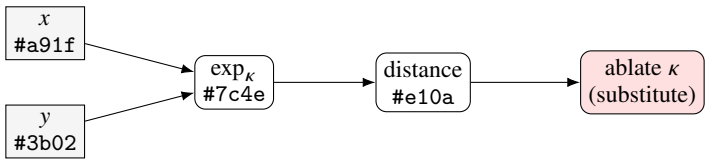

Figure 4: The provenance DAG (Section 6): every primitive call is a node keyed by the content hash of its op, parameters, and input hashes, so identical sub-computations share a node across runs. Substitution at a hex (here zeroing a curvature term) is activation patching expressed on the geometry.

The scope is deliberate. This is a record at the *math-primitive* layer, complementary to neural-network hook libraries (TransformerLens, nnsight). Substitution leaves downstream operations for the caller to re-run, and it is not an experiment tracker. What it enables, by design, is a set of interpretability operations expressed directly on the geometry. These are *causal tracing* of a representation back through the primitives that built it; *ablation* by substituting a node's output (zeroing a curvature term, say) and re-running to measure the downstream effect; content-addressed *deduplication and caching* of repeated sub-computations; and feature analysis over the exported record table. These are the designed-for uses. Demonstrating them at scale on a substrate model is downstream work.

7 Discussion

Across the seven results the mathematics was settled and the work lay in the implementation, making a finite-precision, automatically differentiated, GPU-resident computation behave the way the mathematics prescribes. The cases were a multiplicative factor autograd cannot see (Section 3.1), two algebraically identical distance formulas with opposite numerical fates (Section 3.2), a kernel that is faster and more precise written out by hand than assembled by recursion (Sections 4.1, 4.2), an analytic function that autograd can only reach through a three-part safety net (Section 4.3), and a per-point curvature primitive whose autograd-safety is necessary but not sufficient for stable optimization (Section 5.1). The reference-free dual check (Section 3.3) certifies such a substrate where no reference solution exists.

The bitter lesson predicts that hand-engineering gives way to scale and general methods. For geometric structure to be learned directly instead of laundered through language, the geometry it is learned on has to be correct, fast, and differentiable, so that scale and search operate on the structure itself. These results clear hand-engineering from the math layer (the per-project re-derivation, the silent precision loss, the boundary `NaN`s, the curvature frozen at construction time) and leave the layer above to optimization. A non-Euclidean embedding with learnable, per-point, sign-crossing curvature stays exact and scales on commodity GPUs, the same video hardware the bitter lesson runs on, now carrying lossless geometric structure rather than pixels. Whether structure made directly available shortens the route language takes is the open wager.

The empirical evidence comes via controlled identifiability tasks. Large-scale downstream benchmarks belong to the consuming project. The GPU figures come from two consumer cards instead of a datacenter accelerator or a full-scale model. And the emergence thesis of Section 1 motivates the work while the seven substrate results evidence that they may still be useful whichever way the larger wager resolves. These are peer contributions to an active direction [Giovanni et al., 2022, Guo et al., 2025, Gu et al., 2019]. Natural next steps extend the substrate along the same line. They include a closed-form $\mathbb{H}^9$ kernel from the same operator chain, the spectral-collapse diagnostic applied to a learned embedding rather than known lattices, and downstream tasks that exercise per-point curvature at scale.

Reproducibility. In the `holonomy_lib` repository (`https://github.com/Synoros-io/holonomy_lib`), each result is backed by a symbolic verification script (`notes/verification/`), a numerical demonstration with recorded results (`notes/strengthening/`, `notes/validation/`), and a unit/property/comparison test suite in `tests/`. Every numerical constant in the source is derived, marked as a universal invariant, or cataloged in `notes/magic_numbers.md`, and a static audit enforces this in continuous integration.

Correspondence. Inquiries about downstream applications of this substrate, or collaborative research, are welcome at `contact@synoros.io`.

References

P.-A. Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, 2008.

Gregor Bachmann, Gary Bécigneul, and Octavian Ganea. Constant curvature graph convolutional networks. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119 of *PMLR*, pages 486–496, 2020.

E. B. Davies. *Heat Kernels and Spectral Theory*, volume 92 of *Cambridge Tracts in Mathematics*. Cambridge University Press, 1989.

E. B. Davies and N. Mandouvalos. Heat kernel bounds on hyperbolic space and Kleinian groups. *Proceedings of the London Mathematical Society*, s3-57(1):182–208, 1988.

Xingcheng Fu, Jianxin Li, Jia Wu, Qingyun Sun, Cheng Ji, Senzhang Wang, Jiajun Tan, Hao Peng, and Philip S. Yu. ACE-HGNN: Adaptive curvature exploration hyperbolic graph neural network. In *IEEE International Conference on Data Mining (ICDM)*, pages 111–120, 2021.

Francesco Di Giovanni, Giulia Luise, and Michael M. Bronstein. Heterogeneous manifolds for curvature-aware graph embedding. ICLR 2022 Workshop on Geometrical and Topological Representation Learning, 2022. arXiv:2202.01185.

Alexander Grigor'yan. *Heat Kernel and Analysis on Manifolds*, volume 47 of *AMS/IP Studies in Advanced Mathematics*. American Mathematical Society / International Press, 2009.

Alexander Grigor'yan and Masakazu Noguchi. The heat kernel on hyperbolic space. *Bulletin of the London Mathematical Society*, 30(6):643–650, 1998.

Albert Gu, Frederic Sala, Beliz Gunel, and Christopher Ré. Learning mixed-curvature representations in product spaces. In *International Conference on Learning Representations (ICLR)*, 2019.

Zihao Guo, Qingyun Sun, Haonan Yuan, Xingcheng Fu, Min Zhou, Yisen Gao, and Jianxin Li. GraphMoRE: Mitigating topological heterogeneity via mixture of riemannian experts. In *Proceedings of the 39th AAAI Conference on Artificial Intelligence (AAAI)*, 2025. arXiv:2412.11085.

Hermann Karcher. Riemannian center of mass and mollifier smoothing. *Communications on Pure and Applied Mathematics*, 30(5):509–541, 1977.

Max Kochurov, Rasul Karimov, and Serge Kozlukov. geoopt: Riemannian optimization in PyTorch. *arXiv preprint arXiv:2005.02819*, 2020.

Jack Lindsey, Wes Gurnee, Emmanuel Ameisen, et al. On the biology of a large language model. Transformer Circuits, 2025. `https://transformer-circuits.pub/2025/attribution-graphs/biology.html`.

Tomas Mikolov, Wen tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, pages 746–751, 2013.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *International Conference on Learning Representations (ICLR)*, 2023.

Maximilian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Adam Paszke, Sam Gross, Francisco Massa, et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Xavier Pennec. Intrinsic statistics on Riemannian manifolds: Basic tools for geometric measurements. *Journal of Mathematical Imaging and Vision*, 25(1):127–154, 2006.

R. Rammal and G. Toulouse. Random walks on fractal structures and percolation clusters. *Journal de Physique Lettres*, 44(1):L13–L22, 1983.

Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3D deep learning with PyTorch3D. *arXiv preprint arXiv:2007.08501*, 2020.

Ondrej Skopek, Octavian-Eugen Ganea, and Gary Bécigneul. Mixed-curvature variational autoencoders. In *International Conference on Learning Representations (ICLR)*, 2020.

Richard S. Sutton. The bitter lesson. `http://www.incompleteideas.net/IncIdeas/BitterLesson.html`, 2019. Essay, incompleteideas.net.

Richard S. Sutton and Dwarkesh Patel. Richard Sutton — father of RL thinks LLMs are a dead end. Dwarkesh Podcast, recorded interview, 2025. September 2025, `https://www.dwarkesh.com/p/richard-sutton`.

Abraham A. Ungar. *A Gyrovector Space Approach to Hyperbolic Geometry*. Morgan & Claypool, 2008.

John Vaught. After the universe: A research direction for substrate-faithful representations of meaning. Synoros position paper, 2026. `https://synoros.io/resources/paper`.

Chengjie Yu and Feifei Zhao. Heat kernel recurrence on space forms and applications. *arXiv preprint arXiv:1807.05708*, 2018.