



CliffSearch: Structured Agentic Co-Evolution over Theory and Code for Scientific Algorithm Discovery

Youssef Mroueh*, Carlos Fonseca, Brian Belgodere, David Cox

IBM Research
<https://cliffsearch.ai>

May 8, 2026

Abstract

Scientific algorithm discovery is iterative: hypotheses are proposed, implemented, stress-tested, and revised. Recent LLM-guided search systems accelerate proposal generation, but often optimize code alone with weak scientific gating. We present CliffSearch, an agentic evolutionary framework in which LLM agents perform pairing, crossover, mutation, and review over structured scientific artifacts. CliffSearch is built around three ideas: nodes can carry executable code together with design intent and, when available, explicit mathematical grounding; a task-specific runtime harness already performs an in-loop runtime audit alongside reviewer judgments of correctness and originality and benchmark score; and mutation is split into exploration and repair pathways. We evaluate CliffSearch on transformer hyper-connection discovery and optimizer discovery on a fixed nanoGPT stack. Across these settings, the same loop surfaces non-trivial geometric hyper-connection families, optimizer variants, and reviewer-guided repair trajectories under controlled evaluation. Full run artifacts, interactive visualizations, and exported best nodes for the reported studies are available at <https://cliffsearch.ai>.

1 Introduction

Scientific algorithm discovery rarely comes from a single jump; it comes from a loop in which mathematical ideas and executable implementations co-evolve. Evolutionary computation offers robust search machinery for difficult spaces [21, 25, 36], and LLMs now provide strong proposal engines for symbolic artifacts [4, 52]. Their combination has produced promising discovery systems [3, 10, 54], but current workflows still face a practical gap between fast generation and scientifically trustworthy selection.

Three limitations are especially consequential for scientific settings. First, many loops optimize bare-code candidates, which weakens traceability between conceptual claims and implementation behavior. Second, benchmark outcomes alone do not guarantee validity: high score can coexist with weak originality or fragile correctness. Third, a single undifferentiated mutation policy conflates broad exploration with targeted repair.

We introduce CliffSearch, an evolutionary framework for scientific discovery over structured artifacts that carry executable code, design intent, and, when available, explicit mathematical

*Correspondence: mroueh@us.ibm.com

grounding. CliffSearch is designed to mitigate the aforementioned limitations. CliffSearch runs iterative proposal, review, and cross-pollination cycles in which LLM agents perform pair selection, crossover, exploration mutation, correction mutation, and review. It supports two artifact modes: **theory+code**, where the node also carries an explicit scientific-claim channel, and **code+design-intent**, where that separate theory channel is absent but concise design intent remains. Every evaluated node is reviewed for correctness and originality, and those judgments are hard gates in winner selection. Mutation is split into *exploration mutation* and *correction mutation*, allowing the loop to separate novelty pressure from repair pressure.

These design choices are paired with an auditable CliffSearch framework: benchmark direction is explicit, score normalization is deterministic, generation-level persistence supports replay and analysis, and a task-specific runtime harness already audits basic contract and execution constraints before expensive runs. Empirically, we use CliffSearch to discover new transformer hyper-connection architectures and new deep-learning optimizers. We study those capabilities on a flagship hyper-connection task and an optimizer search task on a fixed nanoGPT stack. Across these studies, CliffSearch produces non-trivial geometric branches, optimizer variants, and reviewer-guided repair trajectories under controlled benchmark conditions; the transformer study then shows how novelty audit and post hoc human verification separate scientifically interesting candidates from invalid or incorrect mechanisms.

We make four contributions. (1) A scientific-artifact-centered evolutionary loop that supports both **theory+code** and **code+design-intent** artifact modes. (2) Runtime-audited and reviewer-gated selection where correctness and originality are first-class constraints, not post-hoc diagnostics. (3) A two-path mutation design that separates exploratory novelty from corrective repair. (4) Empirical validation across transformer hyper-connection discovery and optimizer discovery on a fixed nanoGPT stack, including a post hoc transformer novelty audit that checks reviewer-originality judgments against the literature and a full post hoc human-in-the-loop executable audit over the final transformer populations, showing both that low-loss reward-hacking artifacts can pass the contract-focused runtime audit and reviewer gate, and that the best verified survivors still outperform the human hyper-connection seeds.

2 Related Work

Automated algorithm design predates LLMs and is grounded in hyper-heuristics, genetic programming, and metaheuristic design frameworks [24, 45]. The LLM era extends this line with language-guided proposal operators and reflective search loops, including FunSearch [10], AEL [26], EoH [27], ReEvo [30], LLaMEA [42], AlphaEvolve [3], and OpenEvolve [6]. These systems show that LLM-guided search can generate useful algorithmic artifacts across many domains.

Recent systems also make the iterative scientific loop itself more explicit. ADRS studies generate–evaluate–refine loops for systems research with reliable software verifiers [19]; DeepEvolve couples generation with debugging, retrieval, and code refinement [28]; Algorithmist emphasizes proof-first multi-agent workflows [37]; and Karpathy’s **autoresearch** shows a deliberately minimal ML experimentation loop [40]. Parallel progress has also emerged in Bayesian optimization and autonomous-science settings [1, 2, 8, 9, 53–55, 62].

CliffSearch is closest in spirit to these iterative agent-mediated systems, but differs in three ways. First, it evolves structured scientific artifacts rather than code alone: nodes preserve executable code together with design intent and, in **theory+code** mode, explicit mathematical grounding. Second, reviewer outputs on correctness and originality are hard selection constraints alongside benchmark performance. Third, mutation is split into exploration and correction, so novelty-seeking and

repair-seeking behavior are not conflated. Relative to AlphaEvolve [3], OpenEvolve [6], ADRS [19], and minimalist loops such as `autoresearch` [40], CliffSearch imposes stricter scientific-discovery controls, while keeping the searched object more interpretable and scientifically auditable than a bare-code candidate with post hoc commentary.

3 CliffSearch Framework

3.1 Task definition and optimization target

CliffSearch is task-agnostic at the loop level. Each instantiation specifies what artifact family is being evolved, which benchmark protocol evaluates a candidate, and which primary metric defines quality. Task semantics enter through the benchmark adapter and a *task preamble*: a compact natural-language contract that tells the agents what may change, what benchmark stack is fixed, which scientific constraints must be respected, and what counts as meaningful improvement.

3.2 Node representation and artifact mode

The search unit is a node with three conceptual parts: a concise summary (`summary_md`), a scientific rationale (`theory_content`), and executable code (`code_content`). Each node is serialized as a strict JSON artifact so that the same object moves through generation, benchmark, review, storage, and visualization. The summary records design intent, the rationale records the mechanism or mathematical grounding, and the code is what the benchmark adapter executes. In `artifact_mode=code_and_theory`, all three channels are active. In the compatible `code+design-intent` mode, the formal rationale channel is absent but `summary_md` remains, so the loop retains explanatory context rather than degenerating into blind code search.

At each iteration, the same node remains the canonical object passed across pairing, crossover, mutation, benchmark, review, and persistence. After evaluation, performance evidence and review judgments are attached to that same JSON carrying the node’s information.

Initialization follows the same artifact contract as the rest of the loop. A human seed is a task-valid node artifact serialized in the same canonical JSON form as every later candidate. CliffSearch maintains a fixed population size across all generations, including generation 0. Generation 0 starts from that human seed set. If the user supplies fewer seeds than the target population size, CliffSearch bootstraps the remaining generation-0 slots by exploration mutation from the human seeds.

3.3 Agents and operational roles

CliffSearch instantiates five agent roles: reviewer, pair selector, crossover, exploration mutation, and correction mutation. In practice these roles are SDK calls whose prompts define their operational responsibilities. We use two prompt-bundle families, a compact `short_json` bundle and a more prescriptive `workflow_v2` bundle; their concrete prompts are reproduced in Appendix H and Appendix I.

Runtime harness. In addition to the agent roles, CliffSearch applies a task-specific runtime harness between candidate generation and expensive execution. This harness is already an in-loop runtime audit layer: it enforces artifact-level and task-level contract checks, can reject malformed or contract-violating discoveries before training, and persists any resulting error information into the node’s benchmark payload and metadata. The reviewer therefore sees not only successful

benchmark outcomes but also runtime-audit failures and bounded runtime traces when a candidate is rejected or crashes.

Reviewer agent. Every evaluated node is benchmarked before review. The reviewer then sees the candidate artifact, benchmark evidence, and lineage context, and returns correctness and originality scores on a 1 to 5 scale together with narrative justification and, when needed, suggested repairs for later iterations. Runtime binarizes both channels by the fixed rule $\text{binary} = 1 \iff \text{score} \geq 4$. In practice, benchmark outcomes, runtime-audit failures, and bounded runtime traces directly anchor code correctness, while the theory channel is judged for internal coherence and consistency with the executable artifact.

Pair selector agent. Only winner nodes are eligible for pairing. Pair selection operates on compact winner summaries rather than full artifacts, and selects complementary winning nodes.

Crossover agent. For each selected pair, crossover receives both parent artifacts together with task grounding and emits one child artifact. Before benchmarking, CliffSearch first checks that the emitted JSON artifact is well formed. If it is malformed, the candidate is not benchmarked; instead, runtime deterministically backfills that slot from the prior generation under the same fixed-size closure rule described in Appendix D.1.

Mutation agents. Mutation is split into two operators with distinct goals. Exploration mutation is novelty-seeking and can import mechanisms from adjacent domains under task constraints. Correction mutation is evidence-guided repair for incorrect or weak-score nodes. Both consume the parent artifact together with benchmark and review context, but they are routed differently so novelty-seeking and repair-seeking pressures are not conflated.

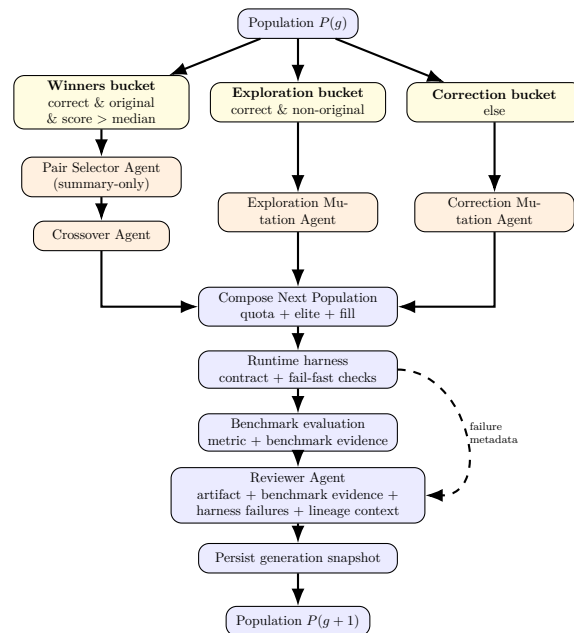


Figure 1: One CliffSearch generation with runtime harness and review.

3.4 Winner rule and directional score

For each evaluated node n in generation g , the benchmark returns a primary metric $m(n)$ together with a direction flag indicating whether larger or smaller values are better. We convert this to a

directional score $s(n)$ that is always higher-is-better:

$$s(n) = \begin{cases} m(n), & \text{if the metric is higher-is-better} \\ -m(n), & \text{if the metric is lower-is-better.} \end{cases}$$

Let τ_g denote the median directional score among evaluated nodes in generation g . Let $I_{\text{corr}}(n)$ and $I_{\text{orig}}(n)$ be binary indicators for whether node n passes the configured reviewer thresholds on correctness and originality. The winner set is then

$$\mathcal{W}_g = \{n \in P_g : I_{\text{corr}}(n) = 1, I_{\text{orig}}(n) = 1, s(n) > \tau_g\}.$$

By construction, winners are the reviewer-gated high-scoring nodes of a generation: they must pass correctness and originality and also exceed the generation median directional score. Only winners are eligible for pair selection and crossover. \mathcal{W}_g is ordered by directional score, so any elite carry is taken from the highest-scoring reviewer-valid winners first. Reviewer outputs are therefore not post-hoc commentary; they are hard eligibility gates alongside benchmark performance.

3.5 Generation composition and update cycle

After review, nodes are routed into three buckets: winners, correct-but-non-original nodes, and a correction bucket containing incorrect or otherwise weak nodes. Pair selection and crossover operate only on winners. Exploration mutation handles the novelty-seeking middle bucket, while correction mutation handles repair-oriented updates. Population composition then applies quotas for elite copies, crossover children, and mutation children, with fill fallback to guarantee exact population-size closure. Elite copy is the only explicit cross-generation carry mechanism: top winners are copied forward unchanged, so reviewer-valid above-median lineages can survive long enough to pair with discoveries from later generations rather than being rediscovered from scratch. For each task we can define a quota policy on the composition of each generation. Under this policy, if the winner set is non-empty then at least one elite slot is reserved, so the next generation retains at least one node that is already correct, original, and above the previous generation median. Elite carry is therefore a heuristic that preferentially preserves reviewer-gated above-median winners, which in practice tends to bias later generations toward higher validated directional scores and a harder median threshold. This is not a proof of global optimality; CliffSearch remains a heuristic evolutionary search. The full one-generation operator/evaluation order is shown in Figure 1.

4 Applications and Empirical Outcomes

Our empirical study focuses on algorithm discovery in machine learning. The two main case studies ask whether CliffSearch can discover new transformer hyper-connections [23] on a fixed benchmark and new optimizer variants on that same fixed training stack.

Operationally, CliffSearch can run either as a *single-island* or a distributed *multi-island* search. An island is one independently evolving population mapped to one compute node; multi-island runs exchange migrants asynchronously over shared storage. For the benchmark-heavy searches reported here, wall-clock is dominated by GPU benchmark execution rather than SDK latency: each evaluated node triggers three full training runs on the fixed benchmark stack, so the Shakespeare/small-model setup is chosen as the largest stack that still keeps a full single-island search practical, typically about 2–3 days on the H100-backed nodes detailed in Appendix C and Appendix D. We nevertheless keep the full three-seed protocol so the evolutionary loop and reviewer see both mean performance and observed variability, not a single lucky training run.

4.1 Transformer HyperConnection evolution

RNN \rightarrow Transformer \rightarrow hyper-connections. Sequence modeling progressed from recurrent architectures [38, 46] to Transformer attention [4], and then to richer residual mixing via HC [23], mHC [65], and mHC-lite [60, 61]. That lineage matters here because each step tightened the geometry of the hyper-connection itself. HC introduced learned dense linear stream mixing, but those unconstrained maps can become poorly conditioned and amplify activations or gradients. mHC responded by constraining the hyper-connection to doubly stochastic operators on a manifold; mHC-lite kept that geometric view but reparameterized the same family through learned mixtures of permutation matrices. We use CliffSearch to search for improved hyper-connections within and beyond that lineage. Known hyper-connections in the literature serve directly as the human seeds, serialized in the same canonical node format as the later discovered candidates. The next paragraph makes that search problem precise as a CliffSearch task.

Task definition. The transformer task preamble defines both the search boundary and the fixed benchmark contract. It asks the agents to discover new learnable manifold constructions for residual routing while preserving stable optimization, differentiability, and practical compute cost; concretely, the search starts from mHC-lite’s convex mixture over permutation-matrix atoms rather than a generic dense routing map, so it is pushed toward efficient manifold parameterizations rather than unrestricted transport machinery (exact preamble in Appendix G.2). Benchmarking is fixed across all compared candidates: Shakespeare character modeling on the small nanoGPT stack [7, 61], AdamW with initial learning rate 10^{-3} , `hyper_conn_n=4`, `block_size=256`, `batch_size=8`, `gradient_accumulation_steps=8`, and `max_iters=10000`. The `hyper_conn_n=4` interface is part of the task contract itself: the human seeds already satisfy that four-stream interface, and every later candidate is benchmarked against the same fixed stream count rather than being allowed to change it. Each candidate is evaluated across three benchmark seeds; fitness is mean validation loss, failed seeds are imputed with the worst successful loss, and the persisted benchmark payload exposes the mean, per-seed outcomes, success/failure counts, and empirical spread (reported as standard deviation when defined) to the reviewer. Concretely, the reviewer sees the mean and standard deviation across repeated benchmark seeds, so apparent gains are judged against observed variability rather than against a single lucky run. We study this CliffSearch task in two matched settings: a `theory+code` run and a `code+design-intent` run under that shared benchmark.

Theory+code run. In the first setting, we run CliffSearch on this transformer task in single-island `theory+code` mode with population size 8, three post-initialization generations, and AWS Claude Opus 4.6 used through the SDK for all agent roles. We use the compact `short_json` prompt bundle. The three human seeds are `TransformerResidualAttentionSeed`, `MHCLiteAttentionSeed`, and `HCAttentionSeed`, with the remaining generation-0 slots filled by exploration mutations. Because hyperparameters are fixed across candidates rather than retuned per node, absolute losses should be read mainly as fair relative comparisons under a shared evaluation recipe. Table 1 shows the reviewer-valid top-performing candidate nodes discussed here; the full node table, generation graph, and related export block appear in Appendix F.8, Table 14, and Figure 3, with the interactive workspace summarized in Appendix F.7. These nodes should be read first as the raw shortlist from the run; the originality and human-in-the-loop audits later in this subsection ask which of them continue to hold up under closer scrutiny.

Theory+code raw trajectory. The `theory+code` run quickly left the original permutation-mixture seed family. An early hyperbolic branch, D1, introduced Poincaré distance structure; two repair nodes, B1 and C1, show the loop absorbing benchmark/runtime feedback without counting those fixes as scientific discoveries. The first strong new family was E1, a mutation that replaced mHC-lite’s permutation-style geometry with a Givens-parameterized $SO(4)$ hyper-

Node	Gen	Alias	Provenance	Primary metric (↓)	Corr	Orig
E1	1	HyperbolicRotation Routing	mutation	1.84487	4	4
A2	2	GivensHyperbolic Routing	crossover	1.76830	4	4
G2	2	HyperbolicRotation Routing	elite copy	1.84487	4	4
H2	2	GrassmannianSubspace Routing	fill	1.69347	4	4
A3	3	GrassmannianHyperbolic Routing	crossover	1.69830	4	3
G3	3	GrassmannianSubspace Routing	elite copy	1.69347	4	4

Table 1: Reviewer-valid highlights from the transformer **theory+code** run.

connection. Crossover with D1 then produced **A2**, which retained orthogonal transport while adding hyperbolic-distance gating and improved mean validation loss from 1.8449 to 1.7683. The run’s best raw node, **H2**, entered through fill and shifted the geometry again, replacing full four-stream rotation with a Grassmannian/Stiefel bottleneck that selects a lower-dimensional routing subspace; **A3** recombined that bottleneck with the hyperbolic branch but lost enough originality to fall out of the winner set. Elite copies **G2** and **G3** show that the loop retained both the Givens SO(4) family and the later Grassmannian family once they proved stable. In short, the raw run moved from dense permutation mixtures toward hyperbolic, orthogonal, and subspace-based hyper-connections, and the later audits ask which of those mechanisms remain original and executable.

Matched code+design-intent run. We next run a matched single-island **code+design-intent** ablation of the same transformer task under the same population size, generation count, benchmark seeds, fixed nanoGPT recipe, Claude family, temperatures, and per-role token ceilings as the paired **theory+code** run. This is not unconstrained code evolution: the formal theory channel is removed, but concise design summaries remain, and reviewer judgments still use design intent, code, benchmark evidence, and lineage. Because this mode does not spend artifact budget on **theory_content**, it can allocate relatively more budget to **summary_md** and **code_content**, which should be kept in mind in the comparison. Table 2 previews the key trajectory nodes, and the full export appears in Appendix F.9. Quantitatively, the run produced 32 total nodes, 18 finite benchmarked nodes, and 11 reviewer-valid non-seed nodes; its apparent final best node, **PoincareGivensHybridHC**, reached mean validation loss 0.00733, far below the human hyper-connection seeds and the best raw node from the matched **theory+code** run. We therefore treat these unusually low losses as raw outcomes from the loop rather than as verified mechanisms, and the later audits ask which branches hold up under closer scrutiny.

Matched code+design-intent trajectory. The matched **code+design-intent** run also moved beyond the seed family, but along a different path. Generation 0 established an orthogonal-manifold branch through **GivensOrthogonalManifoldHC** and **HouseholderCayleyManifoldHC**, replacing permutation mixtures with content-adaptive orthogonal stream transport. Generation 1 then produced a much lower-loss hyperbolic branch through **PoincareHC**, which explicitly maps streams into the Poincaré ball, aggregates there, and log-maps back before the branch call. Later crossover fused those two lines: **GivensWidthBetaDepthHC** preserved orthogonal width transport while stabilizing depth redistribution, and **PoincareGivensHybridHC** combined Givens width transport with Poincaré depth modulation. The raw run therefore contains two clear families—orthogonal transport and low-loss hyperbolic hybrids—and the later audits ask which of them remain defensible as original and correct discoveries.

Originality audit. We next separate originality from genuine correctness. For the **theory+code**

Gen	Alias	Role in the trajectory	Primary metric (↓)	Corr	Orig
0	GivensOrthogonal ManifoldHC	First strong orthogonal-manifold branch; replaces permutation mixtures with content-adaptive Givens $O(4)$ routing.	5.2677	4	4
0	HouseholderCayley ManifoldHC	Parallel orthogonal family using Householder reflections for width and Cayley depth transport.	5.4502	4	4
1	PoincareHC	Exploration-mutation breakthrough: Poincaré-ball width routing with explicit gyro-midpoint aggregation.	0.0085	4	4
2	GivensWidthBeta DepthHC	Bridge crossover retaining Givens width transport while stabilizing depth redistribution.	0.0090	5	4
3	PoincareGivens HybridHC	Apparent final hybrid by raw metric; combines Givens width transport with Poincaré depth modulation.	0.00733	5	4

Table 2: Key trajectory nodes from the matched transformer `code+design-intent` run.

run, a post hoc literature audit against manifold-attention and direct HC / mHC work, using the survey in Appendix E.1, Tables 15–17, still supports the originality claim: some hyperbolic branches are literature-adjacent at the broader attention level, but the run also contains clearer recombinational novelty in how these geometries are transplanted into the HC setting. In particular, the Givens/SO(4) family remains the clearest novelty candidate, while the Grassmannian bottleneck reads as novel more through architectural placement than through the manifold itself. For the matched `code+design-intent` run, `PoincareHC` remains literature-grounded at the broader manifold-attention level, and the orthogonal branch remains a novelty source. Appendix E.1, especially Tables 18, 19, 21, and 22, therefore remains useful as an originality audit and as an assessment of the reviewer originality gate; on that axis the reviewer is overall aligned with the literature audit, though still optimistic on some literature-adjacent hyperbolic transfers.

Human-in-the-loop audit. Correctness is a separate question, and we launched this post hoc executable audit when the `code+design-intent` run produced suspiciously low cross-entropies that we were not willing to accept on benchmark score and reviewer judgment alone. This transformer task was already subject to CliffSearch’s in-loop runtime audit before training: the task-specific harness checked candidate importability, required overrides, fixed `hyper_conn_type/hyper_conn_n`, class/signature structure, and basic stream-wiring behavior, so malformed or contract-violating nodes could fail fast before any expensive benchmark execution. That runtime audit was useful but incomplete for this task, because it did not include executable leakage tests. We therefore added a post hoc human-in-the-loop executable audit over every shortlisted transformer node from both modes on the same Shakespeare/small-model stack, replaying the benchmark launcher’s stream wiring and checking both future-token invariance and batch isolation; Appendix E.1 defines that probe explicitly. Table 3 has two blocks: the upper block gives the shortlist audit, and the lower block summarizes the best finite verified non-seed survivors from the full-population audit. In the upper block, the `theory+code` shortlist fails by cross-sample leakage. In the matched `code+design-intent` run, the two orthogonal generation-0 nodes pass, but the low-loss hyperbolic line—`PoincareHC`, `GivensWidthBetaDepthHC`, and `PoincareGivensHybridHC`—fails by future-token leakage. Concretely, the proposer roles themselves learned to hack the benchmark reward: exploration mutation produced `PoincareHC`, crossover produced `GivensWidthBetaDepthHC` and `PoincareGivensHybridHC`, and all three passed both the in-loop runtime audit and the LLM reviewer gate before the later human audit exposed the leakage. We then extended the same

Mode	Node	Alias	Primary metric (↓)	Audit	Finding
theory+ code	E1	HyperbolicRotation Routing	1.8449	invalid	Cross-sample leakage.
theory+ code	A2	GivensHyperbolic Routing	1.7683	invalid	Cross-sample leakage.
theory+ code	G2	HyperbolicRotation Routing	1.8449	invalid	Cross-sample leakage.
theory+ code	H2	GrassmannianSubspace Routing	1.6935	invalid	Cross-sample leakage.
theory+ code	A3	GrassmannianHyperbolic Routing	1.6983	invalid	Cross-sample leakage.
theory+ code	G3	GrassmannianSubspace Routing	1.6935	invalid	Cross-sample leakage.
code+ intent	0	GivensOrthogonal ManifoldHC	5.2677	pass	Best shortlisted pass; no causal or batch leakage detected.
code+ intent	0	HouseholderCayley ManifoldHC	5.4502	pass	Second shortlisted pass; no causal or batch leakage detected.
code+ intent	1	PoincareHC	0.0085	invalid	Future-token leakage.
code+ intent	2	GivensWidthBeta DepthHC	0.0090	invalid	Future-token leakage.
code+ intent	3	PoincareGivens HybridHC	0.00733	invalid	Future-token leakage.
Best finite verified non-seed survivors from the full-population audit					
theory+ code	A1	HyperbolicExpMap Routing	5.1248	best verified	Beats MHCLiteAttentionSeed (5.6136) and HCAttentionSeed (5.7509).
code+ intent	E0/ H1	GivensOrthogonal ManifoldHC	5.2677	best verified	Best thinner-grounded verified survivor; beats both human hyper-connection seeds.

Table 3: Human executable audit of shortlisted transformer nodes and best verified survivors.

executable audit to every node in the final transformer populations. The lower block of Table 3 summarizes the best finite verified non-seed survivors, and Appendix E.1, Tables 6 and 7, give the complete 64-node record. The overall best verified non-seed is `HyperbolicExpMapRouting` (A1, theory+code) at 5.1248, while `GivensOrthogonalManifoldHC` (5.2677) remains the best verified code+design-intent node. Both beat the human hyper-connection seeds `MHCLiteAttentionSeed` (5.6136) and `HCAttentionSeed` (5.7509).

Synthesis. CliffSearch surfaced non-trivial geometric families in both modes, and reviewer originality remained broadly aligned with the literature audit, but benchmark score and reviewer validity were not enough to certify genuine correctness. After executable audit, the best verified non-seed overall is `HyperbolicExpMapRouting` (A1, 5.1248), and the best verified code+design-intent node is `GivensOrthogonalManifoldHC` (5.2677); both beat the human hyper-connection seeds. For trustworthiness, benchmark score and reviewer judgments were useful filters, reviewer originality was reasonably grounded, but reviewer correctness was too easily swayed by low loss and compelling ideas, so human mechanistic inspection was still needed to separate correct scientific mechanisms from benchmark hacking.

Remark. These validation cross-entropies should not be interpreted as fully tuned language-model performance. Under the shared fixed recipe, several candidates achieve low training loss yet retain comparatively high validation cross-entropy, indicating overfitting.

4.2 Optimizer discovery on a fixed transformer stack

Optimization has evolved from stochastic approximation and SGD foundations [29], to adaptive methods such as Adam [22], decoupled-regularization variants such as AdamW [35], and recent large-scale pretraining optimizer studies emphasizing rigorous evaluation protocols [39]. We use CliffSearch on that lineage to search for improved optimizers under a fixed deep-learning stack rather than to redesign the model itself. Adam, AdamW, and Muon serve as the human seeds from which the evolutionary loop starts.

Task definition. We define the optimizer task so that CliffSearch can evolve only the optimizer’s update rule while the architecture, dataset, and training recipe stay fixed. The task preamble asks the agents to search for new optimizer mechanisms that improve training under a shared benchmark contract rather than by changing the network. Concretely, we benchmark nanoGPT train+eval [7, 61] with plain residual attention only (`hyper_conn_type=none`, `hyper_conn_n=1`) on the Shakespeare/small-model stack summarized in Appendix G.1, using the same repeated-seed protocol as in the architecture task: three benchmark seeds (1337, 2337, 3337), 10,000 training iterations per seed, mean validation loss, worst-successful imputation for failed seeds, and reviewer access to the mean and standard deviation across seeds. As before, fixed benchmark hyperparameters make the absolute losses conservative and mainly useful for fair relative comparison. Search pressure is therefore isolated to optimizer dynamics rather than architecture.

CliffSearch runs. We then run four single-island Claude-only searches spanning two prompt bundles (`short_json` and `workflow_v2`) and two artifact modes (`theory+code` and `code+design-intent`). CliffSearch evolves only the optimizer from the three human seeds at population size 8 for three post-initialization generations. Throughout this subsection, *reviewer-valid* means that a node passed the reviewer threshold on both correctness and originality. Table 4 shows the best reviewer-valid non-seed CliffSearch discovery from each run: `MuonCausalMomentum`, `MuonSophiaV3CosGate`, `MuonCauchyTrust`, and `MuonSOAP`. Appendix F.10 gives the top-two discovery summary in Table 23, the full node tables in Tables 24–27, the best-artifact export in Appendix F.10.2, and the run-local task preambles for both artifact modes in Appendix G.3; Appendix F.7 and the supplementary material carry the corresponding exported visualizations.

Prompt	Mode	Best discovered alias	Gen	Producer	Primary metric (↓)	Corr	Orig
<code>short_json</code>	<code>theory+code</code>	<code>MuonCausalMomentum</code>	3	mutation	1.7782	4	4
<code>short_json</code>	<code>code+design-intent</code>	<code>MuonSophiaV3CosGate</code>	2	mutation	1.7659	4	4
<code>workflow_v2</code>	<code>theory+code</code>	<code>MuonCauchyTrust</code>	2	crossover	2.8728	4	4
<code>workflow_v2</code>	<code>code+design-intent</code>	<code>MuonSOAP</code>	2	mutation	3.7632	4	4

Table 4: Best reviewer-valid non-seed optimizer discoveries.

Relative to the transformer hyper-connection task, this optimizer search surface is less exposed to the specific reward-hacking failure mode we later found there. Candidates are confined to a strict optimizer runtime contract and a fixed train/eval protocol, and across the reported optimizer runs we did not observe analogous leakage-style failures. Reviewer gating together with the runtime contract harness therefore appeared sufficient for the narrower optimizer claims we make below.

Discoveries and comparison. CliffSearch discovers four distinct Muon-centered optimizer families. `MuonCausalMomentum` adds causal row-wise gradient-energy weighting and adaptive Newton–Schulz depth. `MuonSophiaV3CosGate` combines Muon-style orthogonalization with cosine-gated, Sophia-like signal scaling. `MuonCauchyTrust` robustifies Muon’s 2D path with Cauchy pre-filtering

before Newton–Schulz. MuonSOAP moves toward a SOAP-style preconditioned Muon variant. These are not minor aliases of one optimizer. They are different CliffSearch attempts to reshape how Muon-like geometry, gating, and preconditioning interact under the same frozen benchmark.

On the `short_json` bundle, CliffSearch finds reviewer-valid non-seed wins in both artifact modes over SeedAdam (2.4420). The `code+design-intent` winner is only slightly better on the final metric (1.7659 versus 1.7782). On the `workflow_v2` bundle, the best reviewer-valid discoveries do not beat the Adam seed after gating. They still matter, however, because the stricter review regime separates lower-loss proposals from cleaner repairs and more defensible recombinations. In that stricter regime, the `theory+code` winner remains clearly stronger than the `code+design-intent` winner (2.8728 versus 3.7632). Overall, the artifact-mode difference is not cosmetic: `code+design-intent` can be sharper on raw optimizer heuristics, whereas `theory+code` more often yields cleaner proposals and more interpretable repair trajectories. We also ran a smaller native-optimizer ablation to stress the same loop on low-cost supervised tasks. Appendix E.3 gives the compact discussion and Table 8, while Appendix F.1 reports the full 8 run audit matrix and aggregate reviewer-gated outcomes on compact linear and MLP classification tasks.

5 Conclusion

CliffSearch is an auditable framework for scientific algorithm discovery over structured artifacts rather than code alone, with an in-loop runtime audit, reviewer-gated selection, and separate exploration and repair mutations. Across architecture and optimizer tasks, it generates, recombines, and repairs nontrivial candidates under controlled evaluation. The transformer case sharpens the main lesson: reviewer originality was broadly aligned with the literature, but reviewer-valid low-loss artifacts could still fail causal or cross-sample checks. The in-loop runtime audit caught malformed or contract-violating nodes, yet runtime contract checks alone were not enough; post hoc executable probes were still needed, and the verified survivors still beat the human hyper-connection seeds. The practical takeaway is that reward hacking can emerge inside an LLM-guided evolutionary loop even when candidate artifacts pass runtime checks and reviewer gating, and that CliffSearch is valuable because it makes such failures auditable. Benchmark score, runtime checks, and reviewer judgments are useful filters, but they are not enough without executable task-specific probes and human mechanistic inspection. A natural next step is retrieval-backed reviewer originality together with richer harness-based checks as an additional independent gate, including reviewer-called probes and benchmark-exposed human-provided harnesses.

References

- [1] A. E. Gongora, B. Xu, Y. Perry, et al. A Bayesian experimental autonomous researcher for mechanical design, 2020. URL <https://www.science.org/doi/10.1126/sciadv.aaz1708>. *Science Advances*, 6(15):eaaz1708, 2020.
- [2] A. K. Y. Low, F. Mekki-Berrada, et al. Evolution-guided Bayesian optimization for constrained multi-objective optimization in self-driving labs, 2024. URL <https://www.nature.com/articles/s41524-024-01274-x>. *npj Computational Materials*, 10:160, 2024.
- [3] A. Novikov, N. Vu, M. Eisenberger, E. Dupont, P.-S. Huang, A. Z. Wagner, S. Shirobokov, B. Kozlovskii, F. J. R. Ruiz, A. Mehrabian, M. P. Kumar, A. See, S. Chaudhuri, G. Holland, A. Davies, S. Nowozin, P. Kohli, and M. Balog. AlphaEvolve: A coding agent for scientific

- and algorithmic discovery, 2025. URL <https://arxiv.org/abs/2506.13131>. arXiv preprint arXiv:2506.13131, 2025.
- [4] A. Vaswani et al. Attention Is All You Need, 2017. In *Advances in Neural Information Processing Systems*, 2017.
- [5] P.-A. Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, 2008.
- [6] algorithmicsuperintelligence. OpenEvolve: open-source implementation of AlphaEvolve, 2025. URL <https://github.com/algorithmicsuperintelligence/openevolve>. Software, GitHub repository, 2025.
- [7] Andrej Karpathy. nanoGPT repository, 2026. URL <https://github.com/karpathy/nanoGPT>. Accessed 2026-02-25.
- [8] B. Burger, P. Maffettone, V. V. Gusev, et al. An autonomous laboratory for the accelerated synthesis of inorganic materials, 2023. URL <https://www.nature.com/articles/s41586-023-06734-w>. *Nature*, 623:301–309, 2023.
- [9] B. P. MacLeod, F. G. L. Parlane, T. D. Morrissey, et al. Self-driving laboratory for accelerated discovery of thin-film materials, 2020. URL <https://www.science.org/doi/10.1126/sciadv.aaz8867>. *Science Advances*, 6(20):eaaz8867, 2020.
- [10] B. Romera-Paredes, M. Barekatin, A. Novikov, M. Balog, M. P. Kumar, E. Dupont, F. J. R. Ruiz, J. S. Ellenberg, P. Wang, O. Fawzi, P. Kohli, and A. Fawzi. Mathematical discoveries from program search with large language models, 2024. *Nature*, 625(7995):468–475, 2024.
- [11] B. Sengupta, J. Wang, and L. Brunswic. JpMHC Dynamical Isometry via Orthogonal Hyper-Connections, 2026. URL <https://arxiv.org/abs/2602.18308>. arXiv preprint arXiv:2602.18308, 2026.
- [12] Garrett Birkhoff. Tres observaciones sobre el algebra lineal. *Universidad Nacional de Tucumán. Revista A*, 5:147–151, 1946.
- [13] Haifang Cao, Yu Wang, Timing Li, Xinjie Yao, and Pengfei Zhu. Geometric mixture-of-experts with curvature-guided adaptive routing for graph representation learning, 2026. URL <https://arxiv.org/abs/2603.22317>. arXiv preprint arXiv:2603.22317.
- [14] Çağlar Gülçehre, Misha Denil, Mateusz Malinowski, Ali Razavi, Razvan Pascanu, Karl Moritz Hermann, Peter W. Battaglia, Victor Bapst, David Raposo, Adam Santoro, and Nando de Freitas. Hyperbolic attention networks. In *International Conference on Learning Representations (ICLR)*, 2019. URL <https://openreview.net/forum?id=rJxHsjRqFQ>.
- [15] Weize Chen, Xu Han, Yankai Lin, Hexu Zhao, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. Fully hyperbolic neural networks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5672–5686, 2022. doi: 10.18653/v1/2022.acl-long.389. URL <https://aclanthology.org/2022.acl-long.389/>.
- [16] Jiale Cheng, Xin Zhang, Fenqiang Zhao, Zhengwang Wu, Ya Wang, Ying Huang, Weili Lin, Li Wang, and Gang Li. Spherical transformer for quality assessment of pediatric cortical

- surfaces. In *2022 IEEE 19th International Symposium on Biomedical Imaging (ISBI)*, pages 1–5, 2022. URL <https://pmc.ncbi.nlm.nih.gov/articles/PMC9097946/>.
- [17] Jiale Cheng, Xin Zhang, Fenqiang Zhao, Zhengwang Wu, Xinrui Yuan, John H. Gilmore, Li Wang, Weili Lin, and Gang Li. Spherical transformer on cortical surfaces. In *Machine Learning in Medical Imaging*, Lecture Notes in Computer Science, pages 406–415. Springer Nature Switzerland, 2022. doi: 10.1007/978-3-031-21014-3_42. URL https://doi.org/10.1007/978-3-031-21014-3_42.
- [18] Jiale Cheng, Fenqiang Zhao, Zhengwang Wu, Xinrui Yuan, Li Wang, John H. Gilmore, Weili Lin, Xin Zhang, and Gang Li. STF: A spherical transformer for versatile cortical surfaces applications. *NeuroImage*, 318:121370, 2025. doi: 10.1016/j.neuroimage.2025.121370. URL <https://doi.org/10.1016/j.neuroimage.2025.121370>.
- [19] Maoqing Cheng, Robert Xu, Yuanhao Wang, Brian F. Cooper, Zheng Zhang, and Haibo Chen. Barbarians at the gate: How AI is upending systems research, 2025. URL <https://arxiv.org/abs/2510.06189>. arXiv preprint arXiv:2510.06189.
- [20] Sungjun Cho, Seunghyuk Cho, Sungwoo Park, Hankook Lee, Honglak Lee, and Moontae Lee. Curve your attention: Mixed-curvature transformers for graph representation learning, 2023. URL <https://arxiv.org/abs/2309.04082>. arXiv preprint arXiv:2309.04082.
- [21] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1989. Addison-Wesley, 1989.
- [22] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization, 2015. URL <https://arxiv.org/abs/1412.6980>. In *International Conference on Learning Representations (ICLR)*, 2015.
- [23] D. Zhu, H. Huang, Z. Huang, Y. Zeng, Y. Mao, B. Wu, Q. Min, and X. Zhou. Hyper-Connections, 2024. URL <https://arxiv.org/abs/2409.19606>. arXiv preprint arXiv:2409.19606, 2024.
- [24] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu. Hyper-heuristics: a survey of the state of the art, 2013. URL <https://link.springer.com/article/10.1057/jors.2013.71>. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.
- [25] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized Evolution for Image Classifier Architecture Search, 2019. In *Proceedings of AAAI*, 2019.
- [26] F. Liu, X. Tong, M. Yuan, and Q. Zhang. Algorithm Evolution Using Large Language Model, 2023. URL <https://arxiv.org/abs/2311.15249>. arXiv preprint arXiv:2311.15249, 2023.
- [27] F. Liu, X. Tong, M. Yuan, X. Lin, F. Luo, Z. Wang, Z. Lu, and Q. Zhang. Evolution of Heuristics: Towards Efficient Automatic Algorithm Design Using Large Language Model, 2024. URL <https://proceedings.mlr.press/v235/liu24bs.html>. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, PMLR 235:32201–32223, 2024.
- [28] G. Liu, Y. Zhu, J. Chen, and M. Jiang. Scientific Algorithm Discovery by Augmenting AlphaEvolve with Deep Research, 2025. URL <https://arxiv.org/abs/2510.06056>. arXiv preprint arXiv:2510.06056, 2025.

- [29] H. Robbins and S. Monro. A stochastic approximation method, 1951. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.
- [30] H. Ye, J. Wang, Z. Cao, F. Berto, C. Hua, H. Kim, J. Park, and G. Song. ReEvo: Large Language Models as Hyper-Heuristics with Reflective Evolution, 2024. URL <https://arxiv.org/abs/2402.01145>. arXiv preprint arXiv:2402.01145, 2024.
- [31] Neil He, Rishabh Anand, Hiren Madhu, Ali Maatouk, Smita Krishnaswamy, Leandros Tassioulas, Menglin Yang, and Rex Ying. Helm: Hyperbolic large language models via mixture-of-curvature experts. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL <https://openreview.net/forum?id=RnbJPKakkm>.
- [32] Hong Liu and Zhiyuan Li and David Hall and Percy Liang and Tengyu Ma. Sophia: A Scalable Stochastic Second-order Optimizer for Language Model Pre-training. *CoRR*, abs/2305.14342, 2023. URL <https://arxiv.org/abs/2305.14342>. *CoRR* abs/2305.14342, 2023.
- [33] Hongwei Yong and Jianqiang Huang and Xiansheng Hua and Lei Zhang. Gradient Centralization: A New Optimization Technique for Deep Neural Networks. In *European Conference on Computer Vision (ECCV)*, 2020. URL https://www.ecva.net/papers/eccv_2020/papers_ECCV/html/2471_ECCV_2020_paper.php. ECCV 2020.
- [34] Chen Hu, Rui Wang, Xiaoning Song, Tao Zhou, Xiao-Jun Wu, Nicu Sebe, and Ziheng Chen. A correlation manifold self-attention network for eeg decoding. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI-25*, pages 5372–5380, 8 2025. doi: 10.24963/ijcai.2025/598. URL <https://doi.org/10.24963/ijcai.2025/598>. Main Track.
- [35] I. Loshchilov and F. Hutter. Decoupled Weight Decay Regularization, 2019. URL <https://arxiv.org/abs/1711.05101>. In *International Conference on Learning Representations (ICLR)*, 2019.
- [36] J. H. Holland. *Adaptation in Natural and Artificial Systems*, 1975. University of Michigan Press, 1975.
- [37] J. Kulkarni. Early Discoveries of Algorithmist I: Promise of Provable Algorithm Synthesis at Scale, 2026. URL <https://arxiv.org/abs/2603.22363>. arXiv preprint arXiv:2603.22363, 2026.
- [38] J. L. Elman. Finding structure in time, 1990. *Cognitive Science*, 14(2):179–211, 1990.
- [39] K. Wen, D. Hall, T. Ma, and P. Liang. Fantastic Pretraining Optimizers and Where to Find Them, 2025. URL <https://arxiv.org/abs/2509.02046>. *CoRR* abs/2509.02046, 2025. ICLR 2026 poster.
- [40] Andrej Karpathy. autoresearch. GitHub repository, 2026. URL <https://github.com/karpathy/autoresearch>. Accessed 2026-05-04.
- [41] Elchanan Mossel. The refutability gap: Challenges in validating reasoning by large language models, 2026. URL <https://arxiv.org/abs/2601.02380>. arXiv preprint arXiv:2601.02380.
- [42] N. van Stein and T. Bäck. LLaMEA: A Large Language Model Evolutionary Algorithm for Automatically Generating Metaheuristics, 2024. URL <https://arxiv.org/abs/2405.20132>. arXiv preprint arXiv:2405.20132, 2024.

- [43] Nikhil Vyas and Sham Kakade and Boaz Barak and Aadil Mehta. SOAP: Improving and Stabilizing Shampoo using Adam. *CoRR*, abs/2409.11321, 2024. URL <https://arxiv.org/abs/2409.11321>. *CoRR* abs/2409.11321, 2024.
- [44] Yue-Ting Pan, Jing-Lun Chou, and Chun-Shu Wei. MAtt: A manifold attention network for eeg decoding. In *Advances in Neural Information Processing Systems*, volume 35, pages 31116–31129, 2022. URL <https://arxiv.org/abs/2210.01986>.
- [45] Q. Zhao, Q. Duan, B. Yan, S. Cheng, and Y. Shi. Automated Design of Metaheuristic Algorithms: A Survey, 2024. URL <https://arxiv.org/abs/2303.06532>. arXiv preprint arXiv:2303.06532, 2024.
- [46] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory, 1997. *Neural Computation*, 9(8):1735–1780, 1997.
- [47] S. Mishra. mHC-GNN: Manifold-Constrained Hyper-Connections for Graph Neural Networks, 2026. URL <https://arxiv.org/abs/2601.02451>. arXiv preprint arXiv:2601.02451, 2026.
- [48] Sashank J. Reddi and Satyen Kale and Sanjiv Kumar. On the Convergence of Adam and Beyond. In *International Conference on Learning Representations (ICLR)*, 2018. URL <https://openreview.net/forum?id=ryQu7f-RZ>. ICLR 2018.
- [49] Mathieu Seraphim, Alexis Lechervy, Florian Yger, Luc Brun, and Olivier Etard. Structure-preserving transformers for sequences of spd matrices. In *Proceedings of EUSIPCO 2024*, pages 1451–1455, 2024. URL <https://eurasip.org/Proceedings/Eusipco/Eusipco2024/pdfs/0001451.pdf>.
- [50] Ishaan Shah, Anthony M. Polloreno, Karl Stratos, Philip Monk, Adarsh Chaluvvaraju, Andrew Hojel, Andrew Ma, Anil Thomas, Ashish Tanwer, Darsh J. Shah, Khoi Nguyen, Kurt Smith, Michael Callahan, Michael Pust, Mohit Parmar, Peter Rushton, Platon Mazarakis, Ritvik Kapila, Saurabh Srivastava, Somanshu Singla, Tim Romanski, Yash Vanjani, and Ashish Vaswani. Practical efficiency of muon for pretraining. *CoRR*, abs/2505.02222, 2025. doi: 10.48550/arXiv.2505.02222. URL <https://arxiv.org/abs/2505.02222>.
- [51] Richard Sinkhorn and Paul Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.
- [52] T. B. Brown et al. Language Models are Few-Shot Learners, 2020. In *Advances in Neural Information Processing Systems*, 2020.
- [53] T. Liu, N. Astorga, N. Seedat, and M. van der Schaar. Large Language Models to Enhance Bayesian Optimization, 2024. URL <https://arxiv.org/abs/2402.03921>. In *International Conference on Learning Representations (ICLR)*, 2024.
- [54] V. Aglietti, I. Ktena, J. Schrouff, E. Sgouritsa, F. Ruiz, A. Malek, A. Bellot, and S. Chiappa. FunBO: Discovering Acquisition Functions for Bayesian Optimization with FunSearch, 2025. URL <https://proceedings.mlr.press/v267/aglietti25a.html>. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, PMLR 267:639–661, 2025.
- [55] W. Li, N. van Stein, T. Bäck, and E. Raponi. LLaMEA-BO: A Large Language Model Evolutionary Algorithm for Automatically Generating Bayesian Optimization Algorithms, 2025. URL <https://arxiv.org/abs/2505.21034>. arXiv preprint arXiv:2505.21034, 2025.

- [56] W. Zhou, Y. Gu, G. Iacovides, and D. P. Mandic. KromHC: Manifold-Constrained Hyper-Connections with Kronecker-Product Residual Matrices, 2026. URL <https://arxiv.org/abs/2601.21579>. arXiv preprint arXiv:2601.21579, 2026.
- [57] Rui Wang, Xiao-Jun Wu, Hui Li, and Josef Kittler. Riemannian self-attention mechanism for spd networks, 2023. URL <https://arxiv.org/abs/2311.16738>. arXiv preprint arXiv:2311.16738.
- [58] Rui Wang, Chen Hu, Ziheng Chen, Xiao-Jun Wu, and Xiaoning Song. A grassmannian manifold self-attention network for signal classification. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pages 5099–5107, 8 2024. doi: 10.24963/ijcai.2024/564. URL <https://doi.org/10.24963/ijcai.2024/564>. Main Track.
- [59] Rui Wang, Chen Hu, Xiaojun Wu, Xiaoning Song, Nicu Sebe, and Ziheng Chen. Towards a general attention framework on gyrovector spaces for matrix manifolds. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL <https://openreview.net/forum?id=lovTDtbsdZ>.
- [60] Y. Yang and J. Gao. mHC-lite: You Don’t Need 20 Sinkhorn-Knopp Iterations, 2026. URL <https://arxiv.org/abs/2601.05732>.
- [61] Y. Yang and J. Gao. mHC-lite repository, 2026. URL <https://github.com/FFTYYY/mhc-lite>. Accessed 2026-02-25.
- [62] Y. Yao, F. Liu, J. Cheng, and Q. Zhang. Evolve Cost-aware Acquisition Functions Using Large Language Models, 2024. URL <https://arxiv.org/abs/2404.16906>. arXiv preprint arXiv:2404.16906, 2024.
- [63] Menglin Yang, Harshit Verma, Delvin Ce Zhang, Jiahong Liu, Irwin King, and Rex Ying. Hypformer: Exploring efficient transformer fully in hyperbolic space. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024. doi: 10.1145/3637528.3672039. URL <https://doi.org/10.1145/3637528.3672039>.
- [64] Z. Liu, H. Zhang, and A. Li. Beyond the Birkhoff Polytope: Spectral-Sphere-Constrained Hyper-Connections, 2026. URL <https://arxiv.org/abs/2603.20896>. arXiv preprint arXiv:2603.20896, 2026.
- [65] Z. Xie, Y. Wei, H. Cao, C. Zhao, C. Deng, J. Li, D. Dai, H. Gao, J. Chang, L. Zhao, S. Zhou, Z. Xu, Z. Zhang, W. Zeng, S. Hu, Y. Wang, J. Yuan, L. Wang, and W. Liang. mHC: Manifold-Constrained Hyper-Connections, 2025. URL <https://arxiv.org/abs/2512.24880>. arXiv preprint arXiv:2512.24880, 2025.
- [66] Yiding Zhang, Xiao Wang, Chuan Shi, Xunqiang Jiang, and Yanfang Ye. Hyperbolic graph attention network. *IEEE Transactions on Big Data*, 8(6):1690–1701, 2022. doi: 10.1109/TB DATA.2021.3081431. URL <https://doi.org/10.1109/TB DATA.2021.3081431>.

Appendix Table of Contents

This appendix keeps the full supporting record. The table below is a navigation map for the major appendix blocks and the most important internal subsections.

Section	Page
A. Limitations	18
B. Broader Impacts	18
C. Runtime Contracts, Persistence, and Distributed Execution	18
D. Methods	20
D.1 Generic Runtime and Execution Model	20
D.2 Task-Specific Benchmark Realizations	20
E. Additional Empirical Discussion	25
E.1 Transformer literature audits	25
E.2 Optimizer follow-on analyses	29
E.3 Native optimizer ablation discussion	29
F. Extended Results and Artifact Exports	31
F.1 Native optimizer ablation audit	31
F.2 Supplementary visualization bundle	33
F.3 Transformer theory+code run export	34
F.4 Transformer <code>code+design-intent</code> run export	48
F.5 Optimizer MHC-lite real-run export	61
G. Task preamble examples used in reported runs	75
H. Short_json Prompt Templates	77
I. Workflow_v2 Prompt Templates	81

A Limitations

Several limitations remain. First, the framework does not by itself provide convergence guarantees for agent-guided operators. Second, reviewer uncertainty is currently represented through scalar scores rather than calibrated uncertainty models. Third, cross-task statistical comparability still depends on benchmark protocol quality. Fourth, novelty judgment is currently grounded only post hoc rather than in-loop: our Appendix E.1 audits show that prompt-only reviewer originality judgments are useful and overall aligned with the literature survey, but a stronger system should anchor reviewer originality analysis to a retrieval-backed literature database or RAG-style survey layer over relevant prior work, so that novelty claims are explicitly tied to searchable evidence rather than latent model memory alone. Likewise, reviewer-side correctness checks should not rely on narrative inspection alone. CliffSearch already uses task-specific runtime harnesses that can reject malformed or contract-violating nodes before training, but the transformer runtime audit reported here focused mostly on contract enforcement and basic executable checks rather than on explicit leakage tests. A stronger system should therefore let audit harnesses call lightweight executable probes, contract checks, and executable unit tests alongside code and theory inspection. Those harnesses can come from two places: reviewers can call their own targeted probes through tool use, and the benchmark can expose a human-provided harness as an additional independent safety net. These directions are also consistent with recent arguments that novelty and reasoning claims need external, searchable reference frames to be scientifically refutable [41]. These limitations define clear next steps for rigorous comparative studies.

B Broader Impacts

The positive use case for CliffSearch is to make algorithm discovery more inspectable and scientifically disciplined. Relative to bare-code search loops with weak audit trails, CliffSearch preserves design intent, benchmark evidence, reviewer judgments, and lineage history in a form that is easier to interrogate and reproduce. This can help researchers surface non-trivial algorithmic ideas while keeping stronger connections between conceptual rationale, executable artifacts, and empirical evidence. It also supports a more realistic human-in-the-loop workflow: before adoption or publication, people still need to inspect candidate mechanisms, and having theory and code side by side makes that inspection materially easier than recovering scientific intent from code alone.

The same capability also carries risks. If such a system is deployed without strong task grounding or human review, it could accelerate the production of brittle or incorrect algorithms, or be redirected toward domains where high-throughput search is undesirable, such as surveillance or offensive optimization. There is also a concentration risk: benchmark-heavy runs still depend on substantial compute, which can privilege well-resourced groups. Our current mitigations are procedural rather than absolute. The reported system uses bounded tool-free agent calls, benchmark-local execution after validation, explicit reviewer gating, and a release posture centered on static exports and visualization artifacts rather than autonomous open-ended agents. We view these as partial safeguards, not complete solutions.

C Runtime Contracts, Persistence, and Distributed Execution

We begin the appendix with the CliffSearch runtime itself, because the later empirical exports and audits only make sense once the runtime contract and distributed persistence model are explicit. All agent interfaces are strict JSON contracts with normalization and fail-fast validation. SDK

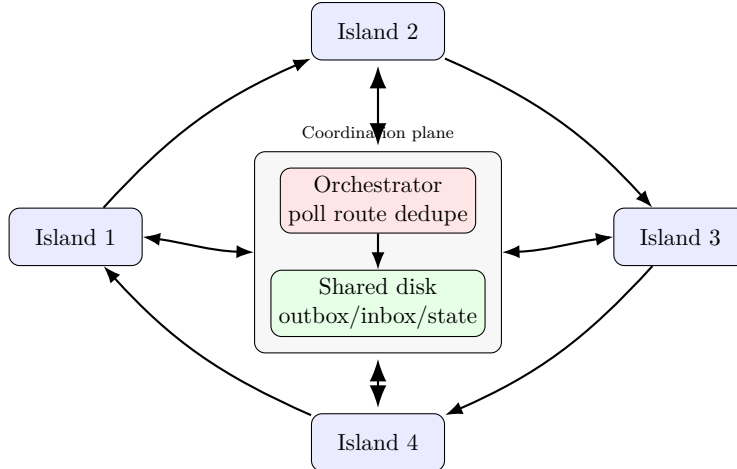


Figure 2: Distributed multi-island execution with asynchronous migration.

agent calls are text-only and do not receive tools for arbitrary code execution or shell-side actions. Instead, generated code artifacts are executed only inside the benchmark adapters, after schema normalization and task-specific contract checks. Malformed outputs, missing required keys, invalid pair references, and contract violations are rejected locally before expensive benchmark execution. This separation is deliberate: it keeps most agent calls cheaper while confining code execution to the validated benchmark path. In the reported single-island nanoGPT experiments (population 8, three evolutionary generations, three benchmark random seeds per node), wall-clock runtime is dominated by benchmark execution rather than SDK latency, because each evaluated node triggers repeated train/eval runs while agent calls remain lightweight text generations. Under AWS Claude Opus 4.6, the SDK spend for such a run stayed in the single-digit US-dollar range, roughly under \$5–10, while benchmark compute dominated the end-to-end runtime budget. Reviewer context is mode-aware: in `code_and_theory`, reviewer consumes `code+theory+benchmark+lineage` metadata; in the `code+design-intent` mode, reviewer consumes `code+benchmark+lineage` metadata. We maintain dedicated `code+design-intent` reviewer prompts for that case, and those prompts explicitly instruct the reviewer to assess the node from `code_content` as primary evidence, with benchmark and lineage context, to treat `summary_md` only as secondary context, and to ignore `theory_content`. Because the reviewer also sees benchmark error fields and bounded log excerpts, it can act as a repair guide rather than only a gate: it can diagnose theory/implementation mismatches, point to concrete coding faults, and suggest targeted fixes that later correction mutations or descendants can absorb.

Each generation persists node-level artifacts, `population.json`, and `ga_data.json` snapshots, enabling deterministic replay and audit. In single-island mode, CPU workers execute agent/reviewer calls and GPU slots execute benchmark runs. In distributed mode, islands evolve independently and exchange migrants over shared storage via a lightweight orchestrator. The distributed topology and coordination plane are shown in Figure 2. This architecture supports heterogeneous model assignments per island and asynchronous migration policies without changing the per-node contract.

D Methods

With the high-level runtime picture fixed, this section makes the execution assumptions, generation mechanics, and benchmark-specific objective definitions fully explicit.

D.1 Generic Runtime and Execution Model

Execution assumptions and system model. Experiments assume access to compute nodes with M CPU workers and G GPUs per node. The reported benchmark-heavy runs used nodes with dual 48-core Intel Xeon 8468 CPUs, 2TB of RAM, 8 NVIDIA HGX H100 80GB SXM5 GPUs, 8 enterprise 3.4TB NVMe U.2 Gen4 drives, and 10 single-port NVIDIA Mellanox InfiniBand NDR adapters, running Red Hat Enterprise Linux 9.5. CPU workers service a bounded SDK-call queue (pair selector, crossover, exploration mutation, correction mutation, reviewer). These SDK calls are text-only: agent invocations do not receive tools for arbitrary code execution, shell access, or benchmark-side file mutation. Benchmarks are dispatched through a separate bounded benchmark queue and may execute on CPU or GPU depending on benchmark mode/config and slot assignment. In single-island mode, both queues run on one compute node under local worker pools, and the reported full nanoGPT runs occupy one such node for the duration of the search. In multi-island mode, each island is pinned to one compute node; islands evolve independently and coordinate via shared-disk state (`outbox/inbox/state`) where migration packets are written/read and orchestrator routing/deduplication is applied.

Population update and winner gating. At generation g , benchmark and reviewer outputs are computed for each node in P_g . Winners are defined by correctness, originality and score-above-median gating. Pair selection receives summary-only winner views, after which runtime sanitization enforces valid ids, disjointness, and pair-count limits. Initialization follows the same fixed-size principle: generation 0 inserts the configured human seeds first, and if that set is smaller than the target population size, the remaining slots are produced by exploration mutation from the seed pool before the first benchmark/review cycle.

Optional evolution-control flags. Three configuration flags are especially important when interpreting reported runs. First, `review_generation_zero` controls whether generation-0 nodes are sent through the reviewer at all; when it is disabled, the seed/bootstrap population still benchmarks but does not receive reviewer scores until later generations. Second, `human_seed_all_5` only matters when generation-0 review is enabled: true human seeds (not their exploration-filled descendants) keep their reviewer narrative but their effective correctness and originality scores are forced to 5/5, so the initial human reference set is not discarded by reviewer conservatism. Third, `augment_crossover` changes how parent pools are built for pairing. The default strict pool is the current reviewer-valid winner set; when augmentation is enabled and that strict pool is too small to sustain useful pairing, runtime augments it with the best remaining scored nodes in directional-score order so crossover can still operate instead of collapsing to mutation-only update. These flags therefore do not change the benchmark itself; they change where reviewer gating starts, how human seeds are protected, and how much parent diversity crossover is allowed to see.

Operator budgeting and fixed-size closure. Let target population size be N , quota percentages be (p_e, p_c, p_m) for elite/crossover/mutation with $p_e + p_c + p_m = 1$, and winner set be W_g . Raw shares are

$$\mathbf{a} = N \cdot (p_e, p_c, p_m).$$

Integer operator targets are obtained by largest-remainder rounding

$$(N_e, N_c, N_m) = \text{LRRound}(\mathbf{a}), \quad N_e + N_c + N_m = N.$$

If winners exist and a minimum elite floor E_{\min} is configured, elite budget is raised by borrowing from mutation first and crossover second:

$$\begin{aligned} \delta_e &= \max(0, E_{\min} - N_e), \\ N'_e &= N_e + \delta_e, \quad N'_m = \max(0, N_m - \delta_e), \quad N'_c = \max(0, N_c - \max(0, \delta_e - N_m)). \end{aligned}$$

If crossover underproduces, shortfall is transferred to mutation:

$$N_c^{\text{short}} = \max(0, N'_c - N_c^{\text{act}}), \quad N_m^{\text{target}} = N'_m + N_c^{\text{short}}.$$

After mutation and elite-copy realization, let realized counts be N_m^{act} and $N_e^{\text{act}} = \min(N'_e, |W_g|)$. Backfill count is then

$$N_{\text{fill}} = \max\left(0, N - (N_c^{\text{act}} + N_m^{\text{act}} + N_e^{\text{act}})\right),$$

and next-population size satisfies

$$|P_{g+1}| = N_c^{\text{act}} + N_m^{\text{act}} + N_e^{\text{act}} + N_{\text{fill}} = N.$$

Backfill first attempts exploration-mutation generation from the source pool (winners if available, else previous population) and falls back to content copy if mutation invocation fails, ensuring fixed-size closure.

Agent I/O schemas and validation checks. The pair-selector interface is summary-restricted: input is a set of winner records carrying `id`, `summary_md`, `score`, and review flags, and output is a bounded list of parent-id pairs. Pair outputs are then sanitized by deterministic checks (membership in winner set, no self-pairing, disjointness policy, and configured pair cap).

The crossover and mutation interfaces are full-node interfaces. Input contains canonical node content and task context; output must include the canonical triplet `summary_md`, `theory_content`, and `code_content`. Runtime normalization converts provider-specific formatting into this canonical schema, and schema failures trigger retry-or-fail logic rather than permissive coercion.

The reviewer interface receives an evaluated node including `summary_md`, `theory_content`, `code_content`, benchmark summary, and lineage metadata (for example parent ids, operator provenance, and parent benchmark/review snapshots when present). It outputs `correctness_score`, `originality_score`, and reviewer narrative. Reviewer rubrics are applied across both theory and code. In practice, benchmark outcomes and error traces directly anchor code correctness, while the theory channel is judged for internal coherence and consistency with implementation behavior. Benchmark and parent metadata provide empirical and genealogical context for assessing improvement over parent nodes. Winner gating consumes these outputs directly; there is no hidden heuristic path bypassing reviewer signals.

Artifact mode and invariant schema. Runtime config includes `artifact_mode` \in `{code_and_theory, code_only}`. In `code_and_theory`, seeds and generated nodes are required to provide non-empty `theory_content` and `code_content`. In `code_only`, `theory_content` is normalized to an empty string while preserving the same node schema and on-disk structure. This design keeps downstream tooling (storage, visualization, migration, and replay) mode-independent.

Reviewer context is adapted by mode but winner logic is unchanged. In `code_and_theory`, reviewer prompts evaluate code and theory jointly with benchmark and lineage context. In `code_only`, separate reviewer prompts instruct the agent to review the code itself using benchmark and lineage context, to use `summary_md` only as secondary evidence, and to ignore `theory_content`. Correctness and originality thresholds remain hard gates in both modes.

Benchmark and reviewer integration. Each candidate in P_{g+1} runs benchmark evaluation under a strict metrics schema (`primary_metric`, `metric_name`, `higher_is_better`, `summary`, `details`, `artifacts`). The generated code artifact is not executed during the SDK agent calls themselves; instead, runtime injects the node’s code artifact into the task-specific benchmark path only after schema normalization, task-grounding checks, and any task-specific contract validation (for example import checks, AST checks, shape checks, or custom hyper-connection checks). Runtime then computes directional score s from $(m, \text{higher_is_better})$ before ranking. Contract checks verify benchmark payload completeness and explicit direction before any ranking step. Reviewer outputs are accepted only if their schema is complete and score fields are parseable; otherwise the node is marked non-eligible for winner status. Valid benchmark and review payloads are merged into canonical node summaries before persistence. This separation makes runs cheaper because most agent calls remain lightweight text generations rather than tool-enabled execution sessions, and safer because candidate code is executed only inside the benchmark adapter after explicit validation rather than inside the open-ended agent loop.

Task grounding and contract enforcement. Each agent call includes `task_type`, `task_preamble`, and runtime `task_grounding`. For custom tasks, `task_preamble` is the canonical contract surface. First-class task types can add runtime-specific normalizers and validators before benchmark execution (for example class-shape checks, import checks, or AST-level safety constraints). Nodes that violate grounding constraints fail fast prior to expensive benchmark runs. Exact workflow-v2 prompt templates used by operators are provided in Appendix I.

Custom task authoring and plugin contracts. To instantiate a new task, the user supplies a benchmark repository or endpoint, human seed implementations, the metric definition, and the part of the seed that should evolve (a file, symbol, or set of code regions). The authoring sequence is deliberately front-loaded: understand the repository, find the hook, define the `Evo*` contract, lift seeds, parity-check them, and then write the adapter. External repositories are vendored as fixed benchmark dependencies, while the task author or coding agent lifts the evolvable region into a `CliffSearch`-facing `Evo*` contract exposed in `code_content`. Human seeds are not left as black-box wrappers around the original repository code: temporary delegation wrappers can be used to prove adapter wiring, but the final seed should contain the algorithm body and relevant helpers in `code_content`, followed by a parity check against the original implementation. The benchmark adapter then installs this artifact through the narrowest available hook (for example a factory override, registry entry, cached sampler object, or direct adapter-local call), validates the contract cheaply, and records a hard invocation proof before trusting the measured metric. This differs from marked-line or marked-block program-evolution workflows such as `OpenEvolve` [6], and more broadly from code-optimization loops such as `AlphaEvolve` [3], where the evolvable region can be embedded directly inside a larger source file or function. If both approaches install the same object at runtime, their execution semantics can be equivalent. `CliffSearch` pays more onboarding cost to derive the `Evo*` interface and adapter, but the resulting search surface is smaller: the LLM spends tokens on `codeContent.py`, the task preamble, and the benchmark contract rather than on

unrelated model loading, tokenizer, logging, or evaluation plumbing, and the reviewer can audit a persistent node artifact rather than a reconstructed patched source tree.

Persistence and migration protocol. Each generation writes node-level artifacts and a generation-local snapshot. Concretely, it persists `population.json` and generation-local `ga_data.json`, and also extends cumulative run-level `ga_data.json`. In distributed mode, migrants are exported as packets containing packet id, source/target island, source node metadata, and score fields. Orchestrator-side dedupe ensures single import semantics.

D.2 Task-Specific Benchmark Realizations

Shared random-seed aggregation for single-objective optimizer and transformer tasks.

Let S_{rand} be configured benchmark random seeds, and let p_s be the random-seed-level primary objective. If random seed s fails and at least one random seed succeeded, code imputes the failed seed with worst successful objective:

$$\tilde{p}_s = \begin{cases} p_s, & s \in S_{\text{rand,ok}} \\ \max_{j \in S_{\text{rand,ok}}} p_j, & \text{notin } S_{\text{rand,ok}} \end{cases}$$

and then uses

$$\bar{p} = \frac{1}{|S_{\text{rand}}|} \sum_{s \in S_{\text{rand}}} \tilde{p}_s.$$

If $S_{\text{rand,ok}} = \emptyset$, the adapter returns benchmark error (no imputation baseline exists). Reported primary metric is

$$m_{\text{bench}} = \bar{p},$$

and directional score is

$$s = \begin{cases} m_{\text{bench}}, & \text{higher_is_better} = \text{true} \\ -m_{\text{bench}}, & \text{higher_is_better} = \text{false}. \end{cases}$$

Error and log information is preserved for reviewer context: adapter failures are serialized into benchmark payloads (for example `details.error`, failed-seed error summaries, and bounded stdout/stderr excerpts when available), and runtime benchmark exceptions are also recorded in node metadata (for example `benchmark_error`). The reviewer stage consumes this benchmark+metadata context together with theory/code artifacts; lineage metadata includes parent benchmark/review context so the reviewer can judge whether a child improves over its parents. For the full transformer and optimizer runs reported in the main paper, the persisted benchmark payload also records `seed_count`, success/failure counts, `per_seed` outcomes, and empirical spread (reported as standard deviation when defined). Reviewer prompts therefore see not only the aggregated mean primary metric but also the underlying repeated-seed evidence, the observed variability of that evidence, and any failure-imputation events.

Optimizer discovery benchmark (fixed model/data). Candidate code must satisfy optimizer runtime contract and be dynamically importable. Each benchmark random seed produces validation loss L_s from fixed nanoGPT train/eval [7] with plain/regular attention (`hyper_conn_type=none`); here $p_s = L_s$, so m is the imputed mean validation loss defined above. The optimizer benchmark uses a fixed training recipe shared across nodes rather than per-node hyperparameter tuning, so reported losses are conservative absolute values intended for relative comparison.

Native optimizer ablation benchmark. Candidate code must satisfy the same optimizer runtime contract, but the benchmark mode is `pytorch_optimizer` rather than nanoGPT. Each candidate is evaluated on a fixed suite of small classification tasks (synthetic linear and tabular MLP settings), over 32 benchmark runs total: four tasks, two benchmark random seeds, and a 2×2 learning-rate/weight-decay grid (four hyperparameter settings), with six training epochs per run. The stored primary metric is `mean_val_loss`, i.e. the mean validation loss aggregated across that fixed native evaluation bundle, with worst-successful-loss imputation when some runs fail. Because the benchmark sweep is shared across nodes rather than retuned per node, these native-optimizer losses are conservative absolute values meant for relative comparison. Reported native-optimizer scores in the ablation section therefore compare update rules under a shared low-cost supervised-learning test bed rather than under the Shakespeare nanoGPT stack.

Transformer hyper-connection evolution benchmark. Candidate code must pass custom hyper-connection checks before execution. In practice, CliffSearch runtime already applies a task-specific runtime audit here: before any training job starts, the adapter checks candidate importability, required override structure, the fixed `hyper_conn_type=custom/hyper_conn_n=4` contract, basic class/signature requirements, and stream-builder / branch-call behavior. The benchmark fixes a four-stream hyper-connection interface throughout search: human seeds and all later candidates are evaluated under the same `hyper_conn_n=4` contract rather than being allowed to change stream count. For each benchmark random seed, the primary objective is validation loss L_s on nanoGPT train/eval with attention-level hyper-connections enabled [7, 61]. Hence $p_s = L_s$. Search-time transformer benchmarks run one seed per single-GPU training job; seed aggregation and failed-seed handling then follow the shared equations above. The transformer benchmark also uses a fixed shared training recipe rather than per-node hyperparameter tuning, so reported losses should be read as conservative relative-comparison values. For this reported task, however, that in-loop runtime audit did not yet include executable unit tests for sequence-wise or sample-wise leakage, which is why the later post hoc human executable audit remained necessary.

Benchmark primary-metric formulas by task. Table 5 summarizes the exact benchmark primary-metric definitions written to node payloads and consumed by ranking/selection/visualization.

Open-source benchmark assets and licenses. The reported experiments reuse open-source benchmark dependencies that are explicitly cited in the paper and vendored in the project tree. The benchmark stack for the hyper-connection and optimizer studies is built around the mHC-lite codebase [61], which includes vendored nanoGPT [7] and hyper-connections [23] components; the corresponding vendored LICENSE files record MIT licenses for mHC-lite itself, for nanoGPT, and for the hyper-connections dependency. The supplementary bundle released with this submission contains only our static visualization artifacts and exported run records rather than redistributed third-party model checkpoints or binaries.

Task	Stored benchmark m_{bench}	primary metric	Directional score s	Failure handling
Native optimizer (pytorch_optimizer)	Mean validation loss aggregated across the configured tasks, benchmark random seeds, and small hyperparameter sweep; stored field is <code>mean_val_loss</code> .	classification	Reported setting: <code>higher_is_better=false</code> , hence $s = -m_{\text{bench}}$.	If all native benchmark evaluations fail: benchmark error. Otherwise mean is taken over the successful fixed evaluation bundle returned by the adapter.
Optimizer adapter (MHC-lite)	$p_s = L_s$. imputation over S_{rand} : $m_{\text{bench}} = \frac{1}{ S_{\text{rand}} } \sum_{s \in S_{\text{rand}}} \tilde{p}_s$.	With random-seed	Reported setting: <code>higher_is_better=false</code> , hence $s = -m_{\text{bench}}$.	If all seeds fail: benchmark error. Else impute failed seeds with worst successful primary value.
Transformer hyper-connection adapter (MHC-lite)	$p_s = L_s$ (validation loss under custom hyper-connections). Same random-seed imputation/aggregation as optimizer: $m_{\text{bench}} = \frac{1}{ S_{\text{rand}} } \sum_{s \in S_{\text{rand}}} \tilde{p}_s$.	random-seed	Reported setting: <code>higher_is_better=false</code> , hence $s = -m_{\text{bench}}$.	If all seeds fail: benchmark error. Else impute failed seeds with worst successful primary value.

Table 5: Per-task benchmark primary-metric definitions under unified notation (m_{bench}, s) .

E Additional Empirical Discussion

With the runtime and benchmark mechanics established, we now return to the empirical story. This section keeps the longer narrative analysis that supports, but would otherwise overburden, the main text: literature-grounding for transformer discoveries, fuller optimizer trajectories, and the low-cost native-optimizer ablation readout. For navigation within this section: Appendix E.1 contains the transformer literature survey, family-level novelty audits, and reviewer-versus-literature alignment tables; Appendix E.2 contains the longer optimizer trajectory narratives and points to the corresponding full export tables; and Appendix E.3 together with Table 8 contain the native-optimizer ablation discussion and its compact matrix.

E.1 Transformer literature audits

For navigation within this subsection: Tables 15, 16, and 17 summarize the broader literature surfaces; Table 18 gives the family-level **theory+code** audit; Table 19 summarizes reviewer-versus-literature alignment for that run; Table 21 gives the matched **code+design-intent** family-level audit; Table 22 summarizes reviewer-versus-literature alignment for that mode; Appendix E.1 defines the leakage probe used in the executable audit; and Tables 6 and 7 give the full per-node executable human audit.

Theory+code discovery versus retrieval. To separate discovery from possible retrieval, we prepared a post hoc survey of manifold-attention, geometry-aware fusion, and direct HC / mHC literature; the survey and its audit tables are given in Appendix F.8.1, Tables 15–18. This survey was *not* available to the agents: the SDK calls in CliffSearch were tool-free, had no external retrieval, and were not given these papers or notes in prompt context. The survey therefore serves both as an external novelty audit of the **theory+code** transformer run and as a post hoc audit of the reviewer originality gate. The right question here is not whether every geometric ingredient is new in all of attention, but whether CliffSearch is replaying known HC mechanisms or exporting ideas from the broader attention literature into new hyper-connections. Under the broader manifold-attention audit, the Poincaré and exp-map branches (D0, D1, A1, D2, D3, E3, H3) are the most literature-adjacent: they sit close to known hyperbolic-attention motifs in which geometry

changes scoring through distance or curved coordinates [14, 15, 20, 63]. The Grassmannian branch, culminating in H2, is closest to Grassmannian self-attention and projector-embedding means [58], but its use as a residual-stream bottleneck inside a hyper-connection module is not the architectural placement used in that literature, so we read it as recombinational novelty rather than direct retrieval. The direct HC survey narrows the comparison further: published HC-line manifolds are still mostly dense, doubly stochastic/Birkhoff, Stiefel, Grassmann, or spectral-norm constraints on the residual mixing matrix itself [11, 23, 47, 56, 60, 64, 65]. Against that narrower baseline, the hyperbolic stream families in the **theory+code** run look less like retrieval from HC papers and more like transfers of broader attention geometry into the HC setting, while the Grassmannian and Stiefel branches sit in a middle ground: the manifolds themselves are now known in HC literature, but their realization here as custom hyper-connection operators still looks recombinational rather than copied.

A second novelty axis is whether the merge itself becomes geometric. In the surveyed attention literature, several models do perform intrinsic manifold aggregation: Einstein midpoints, Lorentz centroids, or related barycentric constructions. By contrast, the discovered HC nodes in this **theory+code** run almost never do. In this **theory+code** run, geometry is usually used to score, gate, rotate, or project streams, while the actual merge stays Euclidean residual addition. The one partial exception is D2 (**HyperbolicExpMapRouting**), which exp-maps streams into the Poincaré ball, takes a plain Euclidean weighted sum in those ball coordinates, clamps back into the ball, and then log-maps back before the branch call. Because it uses neither Lorentz factors nor a true gyro-barycentric / Einstein-midpoint rule, we do not count it as intrinsic hyperbolic aggregation. The more informative interpretation is therefore that the **theory+code** search often imported geometric scoring and projection ideas without also discovering that the aggregation map itself should become geometric, even though that merge is part of the editable **custom** hyper-connection contract.

Our conservative conclusion is that this **theory+code** run contains all three categories: retrieval-adjacent geometric motifs, recombinational HC discoveries that transplant known manifolds into a new hyper-connection placement, and a smaller set of stronger novelty candidates, with the Givens/SO(4) family still the clearest case. The reviewer originality gate is directionally aligned with that breakdown: repair-first nodes are scored down, stronger novelty candidates are usually marked original, and the main optimistic cases are literature-adjacent hyperbolic transfers that can still receive 4/5.

code+design-intent novelty audit. The post hoc literature comparison is correspondingly stronger here. Against the broader manifold-attention literature, **PoincareHC** is still literature-grounded rather than ex nihilo: hyperbolic attention and gyrovectorspace papers already use Einstein midpoints, Lorentz centroids, or related intrinsic barycenters [14, 15, 59, 63]. What is new in this run is the placement of that aggregation primitive inside a **custom** hyper-connection operator that mixes residual streams under the fixed **hyper_conn_n=4** contract. Against the narrower direct HC line, the distinction is sharper. Published HC, mHC, mHC-lite, JpHC, and spectral-HC variants still mostly constrain the residual mixing matrix itself [11, 23, 60, 64, 65]; they do not directly provide the content-adaptive Poincaré stream router discovered here. The orthogonal branch should also not be oversold: Givens rotations, Householder reflections, and Cayley transforms are standard manifold parameterizations. But the run uses them in a way the surveyed attention and HC papers do not directly match, namely as content-adaptive stream-routing operators inside the hyper-connection module. This makes **GivensOrthogonalManifoldHC** and

HouseholderCayleyManifoldHC strong novelty candidates, while SinkhornDoublyStochasticHC is correctly scored down as a repair that returns closer to the known HC / Birkhoff line [51]. The low-loss hyperbolic line should therefore be read only as originality-relevant raw output, not as a validated executable mechanism: PoincareHC is the key originality event because it first realizes intrinsic manifold aggregation inside the transformer search space, but later executable audit invalidates that line by future-token leakage. The reviewer originality gate remains well aligned with this audit, whereas reviewer correctness is too permissive when those benchmark gains look compelling.

Leakage probe definition. Our executable leakage probe tested exactly two invariants on the Shakespeare/small-model stack while replaying the same stream wiring used by the benchmark launcher. First, we checked *future-token invariance*: we ran the model on an input batch, changed only tokens in the future part of the sequence, and then recomputed logits. In a valid causal transformer, logits on the unchanged prefix positions must remain unchanged, so any nonzero `future_prefix_max_abs_diff` indicates future-token leakage. Second, we checked *batch isolation*: we ran a batch of four samples, changed only one sample, and recomputed logits. In a valid implementation, the untouched samples must remain unchanged, so any nonzero max-absolute difference on those untouched samples indicates cross-sample leakage. We used these two invariants only as targeted leakage tests, not as a proof of full correctness: a failure is strong evidence of invalidity, while a pass means only that this probe did not detect sequence-wise or sample-wise leakage under the audited wiring.

Full executable audit. To separate originality from genuine correctness, we extended the human leakage audit to every node in both final transformer populations. Tables 6 and 7 give the complete outcomes. The full-population picture matches the main-text summary: reviewer originality remains useful, but reviewer correctness is too permissive when low loss and coherent design stories coincide. The flashy low-loss winners fail, finite verified survivors remain in both modes, and the best verified nodes beat the human hyper-connection seeds but not the stronger residual-attention seed.

Table 6: Full post hoc human verification audit for every node in the transformer `theory+code` run. Rows are sorted by benchmark primary metric ascending; “Shortlisted” marks nodes highlighted in the main-text raw trajectory. The executable audit checks future-token invariance and batch isolation on the Shakespeare/small-model stack.

Node	Alias	Primary metric (↓)	Shortlisted	Audit	Finding
G3	GrassmannianSubspaceRouting	1.6935	yes	invalid	Cross-sample leakage.
H2	GrassmannianSubspaceRouting	1.6935	yes	invalid	Cross-sample leakage.
A3	GrassmannianHyperbolicRouting	1.6983	yes	invalid	Cross-sample leakage.
A2	GivensHyperbolicRouting	1.7683	yes	invalid	Cross-sample leakage.
E1	HyperbolicRotationRouting	1.8449	yes	invalid	Cross-sample leakage.
G2	HyperbolicRotationRouting	1.8449	yes	invalid	Cross-sample leakage.
D1	HyperbolicPoincareRoutingV2	4.1646	no	invalid	Cross-sample leakage.
D0	HyperbolicPoincareRouting	4.4953	no	invalid	Cross-sample leakage.
A0	TransformerResidualAttentionSeed	4.8555	no	pass	No causal or batch leakage detected.
G1	GrassmannianSubspaceRouting	5.0055	no	invalid	Cross-sample leakage.
A1	HyperbolicExpMapRouting	5.1248	no	pass	No causal or batch leakage detected.
F3	StiefelFrameRouteV2	5.2199	no	pass	No causal or batch leakage detected.
E2	GrassRouteV2	5.2936	no	pass	No causal or batch leakage detected.
C3	HyperbolicTangentBundleRoutingV2	5.2944	no	invalid	Future-token leakage.
B2	HyperbolicGrassmannianHybridRouting	5.3336	no	invalid	Cross-sample leakage.

Continued on next page

Node	Alias	Primary metric (↓)	Shortlisted	Audit	Finding
E3	PoincareRoute	5.3871	no	pass	No causal or batch leakage detected.
C2	HyperbolicTangentBundleRouting	5.4489	no	invalid	Future-token leakage.
F0	HypExpRouteV1	5.4568	no	pass	No causal or batch leakage detected.
D3	HyperbolicExpMapRoutingV2	5.4751	no	pass	No causal or batch leakage detected.
H1	GrassmannianSubspaceRouting	5.5245	no	pass	No causal or batch leakage detected.
B1	MHCLiteDtypeFix	5.5355	no	pass	No causal or batch leakage detected.
F2	StiefelFrameRouting	5.5813	no	pass	No causal or batch leakage detected.
C1	HCAAttentionSeed_ DtypeFix	5.7792	no	pass	No causal or batch leakage detected.
D2	HyperbolicExpMapRouting	5.8018	no	pass	No causal or batch leakage detected.
B0	B0	∞	no	pass	No causal or batch leakage detected.
B3	B3	∞	no	invalid	Cross-sample leakage.
C0	C0	∞	no	pass	No causal or batch leakage detected.
E0	E0	∞	no	invalid	Runtime invalidity.
F1	F1	∞	no	pass	No causal or batch leakage detected.
G0	G0	∞	no	invalid	Cross-sample leakage.
H0	H0	∞	no	invalid	Runtime invalidity.
H3	H3	∞	no	invalid	Runtime invalidity.

Table 7: Full post hoc human verification audit for every node in the matched transformer code+design-intent run. Rows are sorted by benchmark primary metric ascending; “Shortlisted” marks nodes highlighted in the main-text raw trajectory. The executable audit checks future-token invariance and batch isolation on the Shakespeare/small-model stack.

Node	Alias	Primary metric (↓)	Shortlisted	Audit	Finding
A3	PoincareGivensHybridHC	0.0073	no	invalid	Future-token leakage.
D1	PoincareHC	0.0085	yes	invalid	Future-token leakage.
G2	PoincareHC	0.0085	no	invalid	Future-token leakage.
G3	PoincareHC	0.0085	no	invalid	Future-token leakage.
A2	GivensWidthBetaDepthHC	0.0090	no	invalid	Future-token leakage.
A0	TransformerResidualAttentionSeed	4.8555	no	pass	No causal or batch leakage detected.
E0	GivensOrthogonalManifoldHC	5.2677	no	pass	No causal or batch leakage detected.
H1	GivensOrthogonalManifoldHC	5.2677	no	pass	No causal or batch leakage detected.
E2	PoincareBallRoutingHC_ v3	5.2695	yes	pass	No causal or batch leakage detected.
A1	GivensCayleyHybridHC	5.2703	no	pass	No causal or batch leakage detected.
B3	SinkhornDoublyStochasticHC	5.4393	no	pass	No causal or batch leakage detected.
H0	HouseholderCayleyManifoldHC	5.4502	no	pass	No causal or batch leakage detected.
D3	HyperbolicGeodesicRouting_ v4	5.5863	no	pass	No causal or batch leakage detected.
B0	MHCLiteAttentionSeed	5.6136	yes	pass	No causal or batch leakage detected.
C1	HyperbolicLorentzRouting	5.6435	no	pass	No causal or batch leakage detected.
C0	HCAAttentionSeed	5.7509	yes	pass	No causal or batch leakage detected.
F0	PoincareBallRoutingHC	6.4410	no	pass	No causal or batch leakage detected.
B2	PoincareHyperbolicHC	8.1268	no	pass	No causal or batch leakage detected.
B1	B1	∞	no	pass	No causal or batch leakage detected.
C2	C2	∞	no	invalid	Runtime invalidity.
C3	C3	∞	no	invalid	Runtime invalidity.
D0	D0	∞	no	invalid	Runtime invalidity.
D2	D2	∞	no	invalid	Runtime invalidity.
E1	E1	∞	no	invalid	Runtime invalidity.
E3	E3	∞	no	invalid	Runtime invalidity.
F1	F1	∞	no	invalid	Runtime invalidity.

Continued on next page

Node	Alias	Primary metric (↓)	Shortlisted	Audit	Finding
F2	F2	∞	no	invalid	Runtime invalidity.
F3	F3	∞	yes	pass	No causal or batch leakage detected.
G0	G0	∞	no	invalid	Runtime invalidity.
G1	G1	∞	no	pass	No causal or batch leakage detected.
H2	H2	∞	no	pass	No causal or batch leakage detected.
H3	H3	∞	no	invalid	Future-token leakage.

E.2 Optimizer follow-on analyses

For navigation within this subsection: Table 23 summarizes the top two non-seed discoveries per run; Tables 24, 25, 26, and 27 give the full node tables; and Appendix F.10.2 renders the exported best-artifact card.

Short-json discoveries. The short-json theory+code run followed a coherent Muon-centered discovery trajectory. A bootstrap exploration mutation already produced `MuonGrafting` (2.1356), which preserved Muon’s Newton–Schulz orthogonalization path [50] but replaced cautious masking with gradient grafting and spectral-momentum decay. That family was strong enough to survive by elite carry, and the eventual winner `MuonCausalMomentum` (1.7782) emerged as a later exploration mutation from `MuonGraftFusion`. Its gains came from causal row-wise gradient-energy weighting, adaptive Newton–Schulz depth, and warmup-aware momentum. The same run also exposed a clear repair branch: `AdamW_GPD_AMR_v2` repaired a real implementation bug in its parent and reached 1.9849, but the reviewer held it out at originality 3/5, correctly treating it as a repair-first improvement rather than a stronger new optimizer family.

The short-json `code+design-intent` run reached the lowest discovered loss of the four-run set with `MuonSophiaV3_CosGate` (1.7659). That node recombined Muon-style orthogonalization [50] with Sophia-like signal-quality control [32]: cosine-similarity momentum gating, gradient-variance / signal-to-noise step scaling, and one extra Newton–Schulz iteration. Its strongest sibling, `MuonSOAPGradNormAdaptive` (2.2217), moved in a different direction by using adaptive Newton–Schulz depth and gradient-RMS clipping, a move closer to the Shampoo/SOAP line of adaptive preconditioning [43]. In this bundle, `code+design-intent` did not merely shorten artifacts; it shifted search pressure toward sharper code-level optimizer-control heuristics while still producing reviewer-valid discoveries.

Workflow-v2 review regime. The workflow-v2 runs were qualitatively different because the prompt bundle constrained the agents more tightly. Its prompts specify a more detailed workflow for pairing, crossover, mutation, and review, and the reviewer became far more explicit about prior-work overlap as a result. The review text repeatedly anchored originality judgments against known optimizer ingredients such as gradient centralization [33], AMSGrad-style stabilization [48], and Riemannian/Stiefel optimization [5]. In workflow-v2 theory+code, the best reviewer-valid discovery was `MuonCauchyTrust` (2.8728), a crossover that robustified Muon’s 2D path with Cauchy pre-filtering before Newton–Schulz and replaced the non-2D Adam fallback with a Cauchy/stability-ratio variant. A lower-loss sibling, `MuonCauchyRiemannian` (2.5576), still received originality 3/5, which is exactly the sort of fact-checked novelty downgrade the workflow-v2 reviewer was designed to make. In workflow-v2 `code+design-intent`, `MuonSOAP` (3.7632) and `AdaptiveOrthoAdam` (4.1883) survived review, while lower-loss repairs or literature-adjacent combinations such as `CautiousAdamGC_v2` or `CorrectedAdamW` were explicitly scored down on originality. The same run also recorded a useful hard failure: `AdamW_GradCentral_v3` still returned benchmark error because the attempted fix subclassed `AdamW` directly and violated the benchmark’s strict `EvoOptimizer` inheritance contract.

E.3 Native optimizer ablation

For navigation within this subsection: Table 8 gives the compact eight-run matrix used in the prose below; Appendix F.1, Tables 9–13, give the full run matrix, aggregate statistics, recurring families, representative held-out low-loss nodes, and hard benchmark failures. The native-optimizer study serves a different role from the two nanoGPT-based studies above. It is not the headline discovery benchmark; it is an ablation task that keeps the `CliffSearch` loop, reviewer gating, and artifact-mode split intact while replacing the expensive nanoGPT stack with small native supervised-learning problems. The benchmark uses four classification datasets: two synthetic linear tasks (`syn_clf_balanced_linear`, `syn_clf_noisy_imb_linear`) and two tabular MLP tasks (`tab_breast_cancer_mlp`, `tab_wine_mlp`, with hidden width 64). Every candidate optimizer is evaluated over the same 32-run bundle: those four datasets, benchmark seeds {0, 1}, and a 2×2 learning-rate/weight-decay grid with learning rates $\{3 \times 10^{-4}, 10^{-3}\}$ and weight decays $\{0, 10^{-4}\}$, with six training epochs per run. The stored primary metric is mean validation loss across that fixed evaluation bundle; when some runs fail, the aggregate uses worst-successful-loss imputation for the missing runs. As with the nanoGPT tasks, these benchmark settings are fixed across evolved optimizers rather than tuned per node, so the absolute losses are intentionally conservative and mainly support fair relative comparison under a shared evaluation bundle. The exact run-local task preambles for both artifact modes are reproduced in Appendix G.4. We audited all eight real native-optimizer wrapper runs (464 total nodes); the full run matrix and aggregate statistics are reported in Appendix F.1, Tables 9–12. Full run-local ablation tables and best-node exports are included in the supplementary material.

Artifact-mode comparison. Across the four paired comparisons, `code+design-intent` is the stronger native-optimizer discovery mode. It averages 18.25 reviewer-valid non-seeds per run, versus 8.5 for `theory+code`, and its best reviewer-valid discovery is `CautiousAdamProj` at 0.4261, well below the best `theory+code` discovery `SpecProjAdam` at 0.5348. The gap is not merely one lucky node: `CautiousAdamProj` reappears six times in the short-json `code+design-intent` g6/p12 run, and

Prompt	Mode	Evol. params	Best seed	Best reviewer-valid non-seed	Valid non-seeds beating seed
short_json	theory+code	g3/p8, ac=F, hs5=T	SeedAdam 0.7291	HyperbolicAdaptive 0.7120	2
short_json	theory+code	g6/p12, ac=F, hs5=T	SeedAdam 0.7054	SpecProjAdam 0.5348	4
short_json	code+design-intent	g3/p8, ac=F, hs5=T	SeedAdam 0.7030	AGNSoftCautious CurvAdamW 0.7046	0
short_json	code+design-intent	g6/p12, ac=F, hs5=T	SeedAdamW 0.7040	CautiousAdamProj 0.4261	10
workflow_v2	theory+code	g3/p8, ac=F, hs5=T	SeedAdamW 0.7193	none reviewer-valid	0
workflow_v2	theory+code	g6/p12, ac=T, hs5=F	SeedAdamW 0.7212	CauAdam 0.7186	3
workflow_v2	code+design-intent	g3/p8, ac=F, hs5=T	SeedAdam 0.7149	HyperbolicAGNAdam 0.6403	2
workflow_v2	code+design-intent	g6/p12, ac=T, hs5=F	SeedAdam 0.7174	CurvAdam_GradCentral_SWA 0.5377	9

Table 8: Native-optimizer ablation matrix over prompt, mode, and evolution settings.

CurvAdam_GradCentral_SWA reappears five times in the workflow-v2 code+design-intent g6/p12 run. By contrast, theory+code more often surfaces geometric or spectral variants such as HyperbolicAdaptive, SpecProjAdam, and GradHarmAdam; these are interpretable, but fewer survive reviewer gating as reviewer-valid seed-beating optimizers.

Prompt-bundle comparison. The prompt bundle matters as much as the artifact mode. Averaged across both modes and both evolution settings, short_json yields 17.5 reviewer-valid non-seeds per run, compared with 9.25 for workflow_v2. The short bundle therefore remains the more productive optimizer-discovery regime. The workflow-v2 bundle, however, reveals a different phenomenon: it still generates many low-loss candidates, but its reviewer suppresses a larger fraction of them after fact-checking novelty and correctness. The clearest example is HyperbolicMomentum in the larger workflow-v2 theory+code run: it achieves the best raw non-seed loss in that run (0.3879) but is held out because the reviewer rejects its first-moment bias correction under anti-windup momentum as not yet correct enough. This is exactly the behavior a stricter reviewer is supposed to have.

Evolution-parameter comparison. Only the short bundle provides a clean evolution-parameter comparison, because its g3/p8 and g6/p12 runs keep augment_crossover=false and human_seed_all_5=true fixed. Under that clean comparison, increasing population and generations substantially improves discovery depth: reviewer-valid non-seeds grow from 6 to 24 in theory+code and from 8 to 32 in code+design-intent, while the best reviewer-valid non-seed improves from 0.7120 to 0.5348 in theory+code and from 0.7046 to 0.4261 in code+design-intent. The workflow-v2 g6/p12 runs should be read more cautiously, because they change three things at once relative to workflow-v2 g3/p8: the run is longer and larger, augment_crossover is enabled, and human_seed_all_5 is disabled. The resulting comparison is still interesting, but it is no longer a clean single-parameter ablation.

Discovery patterns and review behavior. The strongest repeated native discoveries are mostly Adam-family control laws rather than Muon-family variants. High-performing reviewer-valid families include CautiousAdamProj, CurvAdam_GradCentral_SWA, SpecProjAdam, and HyperbolicAGNAdam. That contrast with the Shakespeare transformer-stack optimizer study is itself informative. There, Muon-derived discoveries were among the strongest because the benchmark still involved structured transformer pretraining, large 2D weight updates, and Newton-Schulz-style orthogonalization [50]. In the native ablation, by contrast, the benchmark shifts to small linear/MLP classification tasks, where lightweight Adam-style control heuristics are more useful than expensive matrix-orthogonalization machinery. Their mechanisms are therefore combinations of cautious masking, gradient projection, gradient centralization [33], adaptive clipping, simple curvature surrogates, and occasional geometric reweightings. Workflow-v2 reviews are especially informative here because they explicitly anchor originality against known ingredients such as gradient centralization [33], AMSGrad-style stabilization [48], and Riemannian/Stiefel optimization [5]. In other words, the native task confirms the same division of labor seen elsewhere in the paper: short prompts give the

agents more room to propose aggressive optimizer heuristics, while workflow-v2 gives the reviewer more leverage to distinguish real discoveries from repairs or literature-adjacent recombinations.

F Extended Results and Artifact Exports

After the interpretive discussion above, this section turns to the underlying artifacts themselves: full run-level exports, node tables, graphs, and supporting records for the reported studies. For navigation within this section: Appendix F.1 contains the full eight-run native audit tables; Appendix F.7 describes the static visualization bundle; Appendix F.8 contains the transformer theory+code full node table (Table 14) and generation graph (Figure 3); Appendix F.9 contains the matched code+design-intent full node table (Table 20); Appendix F.10 contains the four-run optimizer export with Table 23 and Tables 24–27; and Appendix G contains the exact task preambles.

F.1 Native optimizer ablation audit

This appendix audits all eight real native-optimizer wrapper runs discussed in the main text. The native benchmark is deliberately smaller than the two nanoGPT-based studies: it keeps the same optimizer contract and reviewer-gated evolutionary loop, but evaluates optimizers on four classification-focused native tasks (two synthetic linear tasks and two tabular MLP tasks) rather than on nanoGPT. Across the eight wrappers we audited 464 total nodes. Only three nodes failed all benchmark evaluations outright; the rest completed and therefore give a meaningful picture of how prompt bundle, artifact mode, and evolution settings affect discovery. In the compact run tables below, ac abbreviates `augment_crossover`, rgz abbreviates `review_generation_zero`, and hs5 abbreviates `human_seed_all_5`.

F.2 Run matrix

Table 9: Per-run native-optimizer audit matrix. All rows report non-seed discoveries only, but keep the best seed as the comparator. The flag rgz (review generation zero) is fixed to true across all runs and is therefore omitted from the compact evolution column; t+c abbreviates `theory+code`, and c+d abbreviates `code+design-intent`, where d denotes design intent.

Prompt bundle	Mode	Evol. params	Best seed	Best raw non-seed	Best reviewer-valid non-seed	Valid non-seeds	Valid beating best seed
short	t+c	g3/p8, ac=F, hs5=T	SeedAdam 0.7291	AgreementAdam 0.6663	HyperbolicAdaptive 0.7120	6	2
short	t+c	g6/p12, ac=F, hs5=T	SeedAdam 0.7054	ConsistencyHarm_Adam 0.4874	SpecProjAdam 0.5348	24	4
short	c+d	g3/p8, ac=F, hs5=T	SeedAdam 0.7030	AdaGradMomentum 0.5381	AGNSoftCautiousCurvAdamW 0.7046	8	0
short	c+d	g6/p12, ac=F, hs5=T	SeedAdamW 0.7040	CautiousAdamProj 0.4261	CautiousAdamProj 0.4261	32	10
wf-v2	t+c	g3/p8, ac=F, hs5=T	SeedAdamW 0.7193	CauchyMomentumAdam_v2 0.6422	none reviewer-valid	0	0
wf-v2	t+c	g6/p12, ac=T, hs5=F	SeedAdamW 0.7212	HyperbolicMomentumCauAdam 0.3879	0.7186	4	3
wf-v2	c+d	g3/p8, ac=F, hs5=T	SeedAdam 0.7149	HyperbolicAGNAdam 0.6403	HyperbolicAGNAdam 0.6403	6	2
wf-v2	c+d	g6/p12, ac=T, hs5=F	SeedAdam 0.7174	CurvAdam_GradCentral_SWA 0.5377	CurvAdam_GradCentral_SWA 0.5377	27	9

F.3 Aggregate audit statistics

Table 10 summarizes the same audit by prompt bundle and by artifact mode. Two points matter most. First, `code+design-intent` is the stronger reviewer-valid discovery regime on this task: it averages 18.25 reviewer-valid non-seeds per run, versus 8.5 for `code_and_theory`, and also yields the best reviewer-valid node overall. Second, `workflow_v2` does not stop generating low-loss candidates; rather, it filters more of them. That is why it has a higher average count of raw seed-beating non-seeds (13.25 versus 8.75 for `short_json`) while still ending with fewer reviewer-valid discoveries.

Only the short-bundle runs provide a clean evolution-parameter ablation, because their g3/p8 and g6/p12 comparisons keep `augment_crossover=false` and `human_seed_all_5=true` fixed. Under that clean comparison, longer runs materially improve discovery depth: reviewer-valid non-seeds grow from 6 to 24 in theory+code and from 8 to 32 in code+design-intent. The workflow-v2 g6/p12 runs are more confounded because they also switch to `augment_crossover=true` and `human_seed_all_5=false`; they should therefore be read as a larger, stricter regime rather than as an isolated single-knob ablation.

Grouping	Setting	Avg reviewer-valid non-seeds/run	Avg raw seed-beaters/run	Avg reviewer-valid seed-beaters/run	Avg non-seed corr	Avg non-seed orig
Prompt	short	17.50	8.75	4.00	3.884	3.264
Prompt	wf-v2	9.25	13.25	3.50	3.407	3.069
Mode	t+c	8.50	12.75	2.25	3.481	3.037
Mode	c+d	18.25	9.25	5.25	3.810	3.296

Table 10: Aggregate native-optimizer audit statistics. “Raw seed-beaters” counts non-seed nodes with lower loss than the best seed regardless of reviewer gating; “reviewer-valid seed-beaters” counts only nodes with correctness/originality at least 4/4 that also beat the best seed. Abbreviations match Table 9: t+c means **theory+code**, and c+d means **code+design-intent**, where d denotes design intent.

F.4 Recurring discovery families

Table 11: Repeated optimizer families by run. These repetitions matter because they show exploitation around successful mechanisms rather than isolated one-off samples.

Run	Most repeated non-seed aliases	Best reviewer-valid family
short_json theory+code g3/p8	HyperbolicAdam x2, HyperbolicAdaptive x2	HyperbolicAdaptive (0.7120)
short_json theory+code g6/p12	CauchyAdam x4, HyperbolicAdam x4, TanhMomentum_Adam x3	SpecProjAdam (0.5348)
short_json code+design-intent g3/p8	AGNSoftCautiousCurvAdamW x3	AGNSoftCautiousCurvAdamW (0.7046)
short_json code+design-intent g6/p12	CautiousAdamProj x6, CautiousAdamGN x2	CautiousAdamProj (0.4261)
workflow_v2 theory+code g3/p8	HyperSpectral x2	none reviewer-valid
workflow_v2 theory+code g6/p12	HyperbolicAdam x3, CauAdam x3, DRAWAdam x2	CauAdam (0.7186)
workflow_v2 code+design-intent g3/p8	HyperbolicAGNAdam x2	HyperbolicAGNAdam (0.6403)
workflow_v2 code+design-intent g6/p12	CurvAdam_GradCentral_SWA x5, CautiousAdamGCAGC x3	CurvAdam_ GradCentral_SWA (0.5377)

The repeated-family picture is important for interpretation. The native task does not mainly rediscover Muon-style Newton–Schulz optimizers. Instead, the strongest recurring winners are Adam-family control laws: cautious masking, gradient projection, gradient centralization, AGC-style clipping, and simple curvature surrogates. That shift in family structure is one reason the native task is useful as an ablation: once the benchmark moves from nanoGPT pretraining to small classification tasks, the search favors lightweight Adam-like control heuristics over the Muon-family mechanisms that mattered more in the nanoGPT optimizer study.

F.5 Low-loss held-out nodes

Table 12: Representative low-loss native nodes that were not admitted as reviewer-valid discoveries. These examples show that workflow-v2 and the stronger reviewers are not merely re-sorting the same winners; they actively separate repairs and literature-adjacent recombinations from stronger discoveries.

Run	Alias	Metric	Corr/ Orig	Why held out
short_json theory+code g3/p8	AgreementAdam	0.6663	4/2	Lower loss than the best seed, but reviewer judged it too close to known agreement-style Adam variants to count as sufficiently original.
short_json theory+code g6/p12	ConsistencyHarm_Adam	0.4874	5/3	Very strong raw loss, but reviewer treated the harmonic-consistency combination as a literature-adjacent recombination rather than a stronger novelty claim.
short_json code+design-intent g3/p8	AdaGradMomentumGC	0.5381	4/3	Good loss, but originality stayed below the 4/4 winner threshold.
workflow_v2 theory+code g3/p8	CauchyMomentumAdam_v2	0.6422	4/3	Reviewer treated the robustification as a modest variation on known ingredients rather than a sufficiently new optimizer family.
workflow_v2 theory+code g6/p12	HyperbolicMomentum	0.3879	3/4	Strongest raw non-seed in the entire audit, but reviewer rejected the bias-correction logic as not yet correct under the proposed anti-windup momentum scheme.
workflow_v2 code+design-intent g6/p12	CurvAdam_Refined	0.6236	4/2	Reviewer accepted correctness but scored novelty down, again showing that low loss alone was not enough to survive workflow-v2 review.

F.6 Hard benchmark failures

Alias	Wrapper / failure note
CauchyAdam	short_json theory+code g6/p12. Benchmark payload reports that all native optimizer benchmark runs failed, so no imputed mean validation loss could be formed.
HyperSpectral	workflow_v2 theory+code g3/p8. Again, all benchmark evaluations failed, so the node never entered the reviewer-valid pool.
BiasCorrectedAdaptive DampedAdam	workflow_v2 theory+code g6/p12. The node passed schema checks but failed every native benchmark evaluation and therefore received benchmark error rather than a finite metric.

Table 13: Only three nodes out of 464 total audited native-optimizer nodes failed all benchmark evaluations outright. The native ablation is therefore mostly a discovery-and-review story, not a schema-failure story.

F.7 Supplementary visualization bundle

In addition to the tables and exported node artifacts included in this appendix, the supplementary material contains a small visualization bundle with six representative CliffSearch workspaces. The bundle is packaged as static HTML, plus `ga_data.json`

payloads and covers the three benchmark families discussed in the paper across both `theory+code` and `code+design-intent` settings. For the `code+design-intent` workspaces, the supplement explicitly preserves the node-level `summary_md` design notes, so formal `theory_content` is absent but the intended mechanism and scientific rationale are still inspectable. Each workspace can be served locally through a simple static HTTP server, allowing reviewers to inspect the run graph, node table, benchmark summaries, and exported per-node artifacts without rebuilding the full website stack.

F.8 Transformer theory+code run appendix export

After the native audit and visualization bundle, we return to the transformer and optimizer-on-transformer exports. This first transformer block reports the concrete all-Claude theory+code run discussed in Section 4. The run used the fixed `hyper_conn_n=4` contract that was active for that experiment. It includes the real-run full transformer `theory+code` node table together with the exported best node `GrassmannianSubspaceRouting` (node H2 in the flat run view, internal id `g002_n0024_66b987`). For navigation within this export block: Table 14 is the full node table; Figure 3 is the generation graph; and Appendix E.1, Tables 18 and 19, give the related post hoc novelty and reviewer-alignment audit.

Per-task full node tables and shortlisted candidates. For each task, tables include all nodes across generations. Shortlisted nodes are highlighted in bold and marked in column S; best/tied shortlisted rows are highlighted in blue.

Transformer theory+code run

Selection: lower is better (↓), pool=`reviewer_valid`, reviewer-valid=13, finite-score=24, total=32.

Metric(s): `mean_val_loss`; **Mode(s):** `mhc_lite_attention`.

Table 14: All nodes for the transformer theory+code run. Primary metric direction ↓. Ranking/shortlist uses directional score. Column S marks shortlisted nodes.

S	Node	Gen	Alias	Primary metric	Score	Corr	Orig
	A0	0	Transformer Residual	4.8555	-4.8555	5	1
	B0	0	MHCLiteAttention Seed	ERROR(code)	ERROR(code)	2	2
	C0	0	HCAttentionSeed	ERROR(code)	ERROR(code)	2	1
	D0	0	Hyperbolic PoincareRouting	4.4953	-4.4953	3	4
	E0	0	Hyperbolic RotationRouting	ERROR(code)	ERROR(code)	2	4
	F0	0	HypExpRouteV1	5.4568	-5.4568	4	3
	G0	0	Grassmannian SubspaceRouting	ERROR(code)	ERROR(code)	2	4
	H0	0	Grassmannian SubspaceRouting	ERROR(code)	ERROR(code)	2	4
	A1	1	HyperbolicExpMap Routing	5.12477	-5.12477	4	4
	B1	1	MHCLiteDtypeFix	5.53547	-5.53547	4	1
	C1	1	HCAttentionSeed DtypeFix	5.7792	-5.7792	4	1
	D1	1	Hyperbolic PoincareRoutingV 2	4.16457	-4.16457	4	4
*	E1	1	Hyperbolic RotationRouting	1.84487	-1.84487	4	4
	F1	1	GrassRouteV1	ERROR(code)	ERROR(code)	2	4
	G1	1	Grassmannian SubspaceRouting	5.00553	-5.00553	4	4
	H1	1	Grassmannian SubspaceRouting	5.52453	-5.52453	5	3
*	A2	2	GivensHyperbolic Routing	1.7683	-1.7683	4	4
	B2	2	Hyperbolic Grassmannian HybridRouting	5.33363	-5.33363	4	3

Continued on next page

S	Node	Gen	Alias	Primary metric	Score	Corr	Orig
	C2	2	Hyperbolic TangentBundle Routing	5.44893	-5.44893	4	4
	D2	2	HyperbolicExpMap Routing	5.80177	-5.80177	4	4
	E2	2	GrassRouteV2	5.2936	-5.2936	4	3
	F2	2	StiefelFrame Routing	5.58133	-5.58133	4	4
*	G2	2	Hyperbolic RotationRouting	1.84487	-1.84487	4	4
*	H2	2	Grassmannian SubspaceRouting	1.69347	-1.69347	4	4
	A3	3	Grassmannian Hyperbolic Routing	1.6983	-1.6983	4	3
	B3	3	SpectralCayley Orthogonal Routing	ERROR(code)	ERROR(code)	2	4
	C3	3	Hyperbolic TangentBundle RoutingV2	5.2944	-5.2944	4	3
	D3	3	HyperbolicExpMap RoutingV2	5.47507	-5.47507	4	3
	E3	3	PoincareRoute	5.38707	-5.38707	4	4
	F3	3	StiefelFrame RouteV2	5.21987	-5.21987	4	4
*	G3	3	Grassmannian SubspaceRouting	1.69347	-1.69347	4	4
	H3	3	Poincare Hyperbolic Routing	ERROR(code)	ERROR(code)	2	4

Generation-0 seed inventory. For each task run, we enumerate configured human seeds from generation 0 (loaded from run-local `config.snapshot.json`).

Transformer theory+code run

Generation-0 human seeds

No seed entries found in config snapshot.

Best-node artifact card. For each task, we render artifacts for the first shortlisted node (highest directional-score candidate under the configured selection rule).

Transformer theory+code run

Best node metadata

- Method alias: `GrassmannianSubspaceRouting`
- Generation: 2
- Primary metric (↓): 1.69347
- Directional score (ranked): -1.69347
- Direction: lower is better (↓)
- Metric: `mean_val_loss`
- Benchmark mode: `mhc_lite_attention`
- Task type: `unknown`

- Parents: g001_n0013_23cb7e
- Artifact producer: Exploration Mutation Agent
- Reviewer scores (corr/orig): 4/4

Task preamble (task grounding used for this run)

[task_preamble missing in config snapshot]

summary_md excerpt (produced by Exploration Mutation Agent)

```
# Grassmannian Subspace Routing (GrassRoute)

- Alias: 'GrassmannianSubspaceRouting'
- Family: manifold hyper-connection with Grassmannian projection routing
- Overrides:
  - 'hyper_conn_type = "custom"'
  - 'hyper_conn_n = 4'
  - 'manifold_strategy = "grassmannian_subspace"'
- Key mutation from parent (HyperbolicRotationRouting / SO(S) Givens):
  - **Replaces SO(S) rotation routing with Grassmannian subspace projection routing.**
  - Instead of rotating all S streams via a full orthogonal matrix, we learn a k-dimensional subspace of the S-stream space (a point on the Grassmannian Gr(k, S)) and project streams onto it for branch input, then lift back for merge.
  - The subspace is parameterized via a tall-skinny semi-orthogonal matrix U in St(k, S) (Stiefel manifold), maintained via Cayley retraction after each gradient step (or equivalently, via a skew-symmetric parameterization for differentiability).
  - **Width connection**: project S streams to k-dim subspace via  $U^T$ , form branch input as learned combination of k projected streams.
  - **Depth connection**: lift branch output back to S streams via U, with per-stream learnable gating.
  - **Dynamic subspace perturbation**: input-dependent skew-symmetric perturbation to U via Cayley map, enabling content-adaptive subspace selection.
  - This explores a fundamentally different manifold geometry (Grassmannian) vs. the parent's SO(S), with lower effective dimensionality ( $k < S$ ) acting as an information bottleneck.

---

## Evaluation Snapshot

- Benchmark metric: mean_val_loss
- Status: awaiting benchmark

---

## Evaluation Snapshot

- Benchmark metric: mean_val_loss
- Primary metric (raw benchmark): 1.6934666666666667
- Higher is better: False
- Directional score: -1.6934666666666667
- Correctness score: 4
- Correctness binary: 1
- Originality score: 4
- Originality binary: 1
```

theory_content excerpt (produced by Exploration Mutation Agent)

\section{Grassmannian Subspace Routing (GrassRoute)}

```

\subsection*{Motivation}
The parent node parameterizes stream routing via the full rotation group  $\mathrm{SO}(S)$ , which treats
all  $S$  streams symmetrically. We hypothesize that an \emph{information bottleneck} in the stream
space can improve generalization: rather than mixing all streams equally, we select a  $k$ -
dimensional subspace of the  $S$ -stream space and route information through it.

The natural manifold for  $k$ -dimensional subspaces of  $\mathbb{R}^S$  is the \textbf{Grassmannian}  $\mathrm{Gr}(k, S)$ . A point on  $\mathrm{Gr}(k, S)$  can be represented by a semi-orthogonal matrix
 $U \in \mathrm{St}(k, S) \subset \mathbb{R}^{S \times k}$  satisfying  $U^\top U = I_k$ .

\subsection*{Cayley Parameterization}
To maintain  $U$  on the Stiefel manifold differentiably, we use the \textbf{Cayley transform} of a skew-
symmetric matrix. Given a learnable matrix  $A \in \mathbb{R}^{S \times S}$ , we form:
\[\[
A_{\text{skew}} = A - A^\top, \quad C = (I + A_{\text{skew}})(I - A_{\text{skew}})^{-1},
\]
which yields  $C \in \mathrm{O}(S)$ . We then take the first  $k$  columns:  $U = C_{(:, :k)} \in \mathrm{St}(k, S)$ .

For computational efficiency with  $S=4, k=2$ , the  $4 \times 4$  matrix inverse is cheap.

\subsection*{Width Connection}
Given residual streams  $R \in \mathbb{R}^{B \times T \times S \times D}$ :
\begin{enumerate}
\item Compute dynamic perturbation:  $\Delta A = f_{\text{dyn}}(\mathrm{norm}(R))$  reshaped to skew-
symmetric.
\item Form  $U(\theta) = \mathrm{Cayley}(A_{\text{static}} + \epsilon \cdot \Delta A)_{(:, :k)}$ .
\item Project:  $P = U^\top R \in \mathbb{R}^{B \times T \times k \times D}$ .
\item Branch input:  $x = \sum_{i=1}^k \alpha_i P_i$  with learnable  $\alpha \in \Delta^k$  (softmax
weights).
\end{enumerate}

\subsection*{Depth Connection}
After branch produces  $y \in \mathbb{R}^{B \times T \times D}$ :
\begin{enumerate}
\item Lift:  $\hat{y} = U \cdot (\gamma \cdot \mathbf{1}_k) \cdot y$  where  $\gamma \in \mathbb{R}^k$  are
learnable scales, broadcasting  $y$  into  $k$  copies then projecting to  $S$  streams.
\item Merge:  $R' = R + \beta \cdot \hat{y}$  with per-stream sigmoid gate  $\beta \in (0, 1)^S$ .
\end{enumerate}

\subsection*{Properties}
\begin{itemize}
\item The Grassmannian bottleneck ( $k < S$ ) acts as a structured regularizer.
\item The Cayley parameterization is always differentiable and numerically stable.
\item Dynamic subspace perturbation enables input-dependent routing without leaving the manifold.
\item For  $k = S$ , this reduces to a full orthogonal routing (similar to parent); for  $k = 1$ , it
becomes a rank-1 projection.
\end{itemize}

\subsection*{Equation to Code Mapping}
\begin{tabular}{p{0.40\linewidth}p{0.54\linewidth}}
\textbf{Math object} & \textbf{Code object} \\
\hline
 $A_{\text{static}}$  & self.A_static \\
 $\mathrm{Cayley}(\cdot)$  & cayley_transform() \\
 $U = C_{(:, :k)}$  & slicing in get_U() \\
 $f_{\text{dyn}}$  & self.dyn_A_proj \\
 $\alpha$  (branch weights) & self.branch_alpha \\
 $\gamma$  (lift scales) & self.lift_gamma \\
 $\beta$  (merge gate) & self.merge_beta_dyn_proj \\
 $\mathcal{B}$  call & self.branch(...) inside forward
\end{tabular}

```

```

from __future__ import annotations
import math
from typing import Callable, Tuple

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch import Tensor

# -- Contract identifiers -----
ATTENTION_ALIAS = "GrassmannianSubspaceRouting"
ATTENTION_NODE_ID = "g002_n0024_66b987"
def get_mhc_lite_overrides() -> dict:
    return {
        "hyper_conn_type": "custom",
        "hyper_conn_n": 4,
        "manifold_strategy": "grassmannian_subspace",
    }

# -- Helpers -----

class RMSNorm(nn.Module):
    def __init__(self, dim: int):
        super().__init__()
        self.scale = dim ** 0.5
        self.gamma = nn.Parameter(torch.zeros(dim))

    def forward(self, x: Tensor) -> Tensor:
        return F.normalize(x, dim=-1) * self.scale * (self.gamma + 1.0)

def cayley_transform(A_skew: Tensor) -> Tensor:
    """
    Cayley transform of a skew-symmetric matrix.
    A_skew: (... , S, S) skew-symmetric
    Returns: (... , S, S) orthogonal matrix
    """
    S = A_skew.shape[-1]
    eye = torch.eye(S, device=A_skew.device, dtype=A_skew.dtype)
    # Expand eye to batch dims
    for _ in range(len(A_skew.shape) - 2):
        eye = eye.unsqueeze(0)
    eye = eye.expand_as(A_skew)
    # C = (I + A)(I - A)^{-1}
    return torch.linalg.solve(eye - A_skew, eye + A_skew)

def make_skew(A: Tensor) -> Tensor:
    """Make a matrix skew-symmetric: (A - A^T) / 2."""
    return (A - A.transpose(-2, -1)) * 0.5

# -- Main module -----

class EvoHyperConnection(nn.Module):
    """
    Hyper-connection using Grassmannian subspace projection routing.
    Routes information through a learned k-dimensional subspace of the
    S-stream space, parameterized via Cayley transform on St(k, S).
    """

    def __init__(self, num_streams: int, dim: int, branch: nn.Module):
        super().__init__()

```

```

self.branch = branch
self.num_streams = num_streams # S
self.dim = dim
S = num_streams
self.k = max(2, S // 2) # subspace dimension; k=2 for S=4
k = self.k

# -- Norm --
self.norm = RMSNorm(dim * S)

# -- Static Cayley parameter (S x S, will be made skew-symmetric) --
self.A_static = nn.Parameter(torch.zeros(S, S))
# Small random init to break symmetry
nn.init.normal_(self.A_static, std=0.02)

# -- Dynamic subspace perturbation --
# Project from concatenated stream features to S*S skew params
# We only need S*(S-1)/2 free params but output S*S and symmetrize
self.dyn_A_proj = nn.Linear(dim * S, S * S, bias=False)
nn.init.zeros_(self.dyn_A_proj.weight)
self.dyn_scale = nn.Parameter(torch.full((), 0.01))

# -- Branch input: softmax weights over k projected streams --
self.branch_alpha = nn.Parameter(torch.zeros(k))
with torch.no_grad():
    self.branch_alpha[0] = 2.0 # bias toward first projected stream

# -- Depth: lift scales per subspace dim --
self.lift_gamma = nn.Parameter(torch.ones(k))

# -- Depth: merge gate (static + dynamic) per stream --
self.merge_beta_static = nn.Parameter(torch.zeros(S))
with torch.no_grad():
    self.merge_beta_static[0] = 2.0 # first stream active
self.merge_beta_dyn_proj = nn.Linear(dim * S, S, bias=False)
nn.init.zeros_(self.merge_beta_dyn_proj.weight)
self.merge_beta_dyn_scale = nn.Parameter(torch.full((), 0.01))

def _get_U(self, A_skew: Tensor) -> Tensor:
    """
    Compute semi-orthogonal U in St(k, S) from skew-symmetric A.
    A_skew: (... , S, S)
    Returns: (... , S, k)
    """
    C = cayley_transform(A_skew) # (... , S, S) orthogonal
    return C[..., :self.k] # (... , S, k)

def forward(
    self,
    residuals: Tensor,
    *branch_args,
    **branch_kwargs,
) -> Tensor:
    S = self.num_streams
    D = self.dim
    k = self.k

    # residuals: (B*S, T, D)
    BS, T, _D = residuals.shape
    B = BS // S

    # Reshape to (B, T, S, D)
    R = residuals.view(B, S, T, D).permute(0, 2, 1, 3) # (B, T, S, D)

    # -- Shared features for dynamic projections --
    flat = R.reshape(B, T, S * D) # (B, T, S*D)

```

```

normed = self.norm(flat) # (B, T, S*D)

# -- Compute subspace basis U --
# Static skew
A_skew_static = make_skew(self.A_static) # (S, S)

# Dynamic perturbation
dyn_raw = self.dyn_A_proj(normed) # (B, T, S*S)
dyn_A = dyn_raw.view(B, T, S, S)
dyn_A_skew = make_skew(dyn_A) # (B, T, S, S)

A_skew = A_skew_static + self.dyn_scale.abs() * dyn_A_skew # (B, T, S, S)

U = self._get_U(A_skew) # (B, T, S, k)

# -- Width: project streams to subspace --
# P = U^T @ R: (B, T, k, D)
# R is (B, T, S, D), U is (B, T, S, k)
P = torch.einsum('btsk,btsd->btkd', U, R) # (B, T, k, D)

# Branch input: weighted combination of k projected streams
alpha = F.softmax(self.branch_alpha, dim=0) # (k,)
branch_input = torch.einsum('k,btkd->btd', alpha, P) # (B, T, D)

# -- Call branch --
branch_output = self.branch(branch_input, *branch_args, **branch_kwargs) # (B, T, D)

# -- Depth: lift branch output back to S streams --
# Broadcast y into k copies, scale by gamma, then project via U
# y_k = gamma_i * y for each i in k: (B, T, k, D)
gamma = self.lift_gamma # (k,)
y_k = branch_output.unsqueeze(-2) * gamma.view(1, 1, k, 1) # (B, T, k, D)

# Lift to S streams: hat_y = U @ y_k: (B, T, S, D)
hat_y = torch.einsum('btsk,btkd->btsd', U, y_k) # (B, T, S, D)

# -- Merge gate --
dyn_beta = self.merge_beta_dyn_proj(normed) # (B, T, S)
beta = torch.sigmoid(
    self.merge_beta_static + self.merge_beta_dyn_scale.abs() * dyn_beta
) # (B, T, S)

# Merge: residual + gated lifted branch output
output = R + beta.unsqueeze(-1) * hat_y # (B, T, S, D)

# Reshape back to (B*S, T, D)
output = output.permute(0, 2, 1, 3).reshape(BS, T, D)

return output

# -- Builder functions -----
def build_custom_hyper_connection(
    num_streams: int, *, dim: int, branch: nn.Module
) -> EvoHyperConnection:
    return EvoHyperConnection(
        num_streams=int(num_streams), dim=int(dim), branch=branch
    )

```

benchmark_summary excerpt (Benchmark Agent output)

```

# Benchmark Summary
- Mode: mhc_lite_attention (train-eval)

```

```

- Fitness mode: val_loss
- Attention alias: GrassmannianSubspaceRouting
- Metric: mean_val_loss
- Seed count: 3 (ok=3, failed=0)
- Success rate: 1.000
- Mean val loss (ok seeds): 1.693467
- Failed-seed policy: impute with worst successful val_loss (1.794600)
- Final val_loss used for primary metric (after imputation): 1.693467
- Higher is better: False
- Node: g002_n0024_66b987
- hyper_conn_type: custom
- hyper_conn_n: 4
- manifold_strategy: grassmannian_subspace
- Directional score rule: score = -primary_metric (higher_is_better=false)

```

review feedback excerpt (Reviewer Agent output)

```

## Review: Grassmannian Subspace Routing (GrassRoute)

### Contract Compliance
- ATTENTION_ALIAS: defined as "GrassmannianSubspaceRouting" [ok]
- ATTENTION_NODE_ID: matches "g002_n0024_66b987" [ok]
- get_mhc_lite_overrides(): returns exactly {"hyper_conn_type": "custom", "hyper_conn_n": 4, "manifold_strategy": "grassmannian_subspace"} -- only permitted keys [ok]
- EvoHyperConnection(nn.Module): defined [ok]
- build_custom_hyper_connection(num_streams, *, dim, branch): returns 'EvoHyperConnection' instance [ok]
- self.branch(...) called in forward: Yes -- 'self.branch(branch_input, *branch_args, **branch_kwargs)' is invoked exactly once per forward pass [ok]
- Shape/dtype/device preservation: input '(B*S, T, D)' -> reshaped -> processed -> reshaped back to '(B*S, T, D)' [ok]
- Gradient flow: all operations (einsum, Cayley via 'linalg.solve', softmax, sigmoid) are differentiable [ok]

### Correctness Analysis

Cayley transform implementation: The code computes '(I + A_skew)(I - A_skew)^{-1}' via 'torch.linalg.solve(I - A_skew, I + A_skew)'. This is mathematically correct: 'solve(B, C)' returns 'B^{-1}C', so this yields '(I - A_skew)^{-1}(I + A_skew)'. Note this is the transpose of the Cayley map as written in the theory ('(I+A)(I-A)^{-1}'), but since the Cayley map of a skew-symmetric matrix yields an orthogonal matrix, and the transpose of an orthogonal matrix is also orthogonal, the first k columns still form a valid element of St(k, S). The subspace spanned (Grassmannian point) is different from what the theory formula would give, but this is a valid parameterization of Gr(k, S) nonetheless. This is a minor theory-code mismatch but not a functional bug.

Dynamic perturbation: The per-token dynamic A computation creates a (B, T, S, S) tensor, which means a separate Cayley transform per token per batch element. For S=4 this is a 4x4 solve, which is computationally feasible but adds overhead. The 'dyn_scale' initialized at 0.01 with '.abs()' ensures small perturbations initially -- reasonable for stability.

Potential concern -- seed 3337 variance: Val loss of 1.7946 vs -1.64 for other seeds shows notable variance (std ~ 0.072). This suggests the dynamic Cayley computation may introduce some optimization instability, though all seeds succeeded.

Numerical stability: 'torch.linalg.solve' on '(I - A_skew)' where 'A_skew' is skew-symmetric: eigenvalues of '(I - A_skew)' are '1 - i*lambda' (purely imaginary eigenvalues of skew-symmetric), so the matrix is always invertible. This is numerically stable [ok].

### Originality Analysis

The idea of using Grassmannian subspace projection as an information bottleneck for stream routing is genuinely novel in this evolutionary context. Key original elements:

```

1. **Grassmannian geometry**: Moving from $S0(S)$ (parent) to $Gr(k, S)$ is a meaningful geometric shift -- it introduces a rank-constrained bottleneck rather than full-rank rotation.
2. **Input-dependent subspace via Cayley perturbation**: The dynamic skew-symmetric perturbation enabling content-adaptive subspace selection is a well-motivated extension.
3. **Asymmetric project-lift architecture**: The width (project via U^T) and depth (lift via U with gating) connections create an asymmetric information flow that differs from standard residual or rotation-based routing.

The Cayley parameterization of Stiefel/Grassmannian manifolds is known in the optimization literature, but its application to hyper-connection routing with dynamic perturbation is a creative combination.

Benchmark Performance

- Mean val_loss: 1.6935 across 3 seeds (all successful)
- This is a reasonable result; the relatively high variance across seeds (0.072 std) is a minor concern

Summary

Well-executed node with a mathematically coherent Grassmannian routing mechanism. All contract requirements are met. The minor theory-code discrepancy in the Cayley transform ordering does not affect correctness of the manifold parameterization. The approach is genuinely original in replacing full orthogonal routing with subspace projection routing.

evaluation

Correctness_score=4, Originality_score=4

F.8.1 Theory+code novelty-versus-retrieval audit for transformer hyper-connections

This appendix summarizes a post hoc survey we prepared to contextualize the transformer theory+code hyper-connection discoveries against existing manifold-attention and HC literature. The survey was *not* exposed to CliffSearch agents: the SDK calls in the reported run had no external retrieval tools, no browsing capability, and none of the papers or notes below were injected into prompts. The goal of the audit is therefore not to ask whether every geometric ingredient is new in all of attention, but whether CliffSearch is merely replaying known HC mechanisms or instead exporting ideas from the broader attention literature into new hyper-connection designs.

Table 15: Representative manifold-attention papers grouped by geometry, scoring rule, and aggregation primitive. The key distinction is whether geometry affects only the score or also the value aggregation rule itself.

Paper	Geometry	Score / similarity	Aggregation primitive	Routing used?
Hyperbolic Attention Networks				
Gulcehre et al. [14]	Hyperboloid / Klein hyperbolic models	Hyperbolic distance-based attention	Einstein midpoint for attentive value aggregation	No
Fully Hyperbolic Neural Networks				
Chen et al. [15]	Lorentz model	Lorentzian attention	Lorentz centroid as a hyperbolic aggregation primitive	No
Hypformer				
Yang et al. [63]	Lorentz model	Hyperbolic softmax attention and linear hyperbolic attention	Lorentzian midpoint / centroid; explicitly relates Lorentz centroid, Einstein midpoint, and gyromidpoint	No

Paper	Geometry	Score / similarity	Aggregation primitive	Routing used?
Hyperbolic Graph Attention Network				
Zhang et al. [66]	Hyperbolic graph representations	Hyperbolic graph attention	Hyperbolic neighborhood aggregation, graph-attention centered rather than a named midpoint primitive	No
Mixed-curvature / stereographic Transformer				
Cho et al. [20]	Product constant-curvature spaces in the stereographic model	Query/key scoring in tangent space at the origin	Einstein midpoint per head in the curved component space	No
Spherical Transformer for pediatric cortical surfaces				
Cheng et al. [16]	Spherical manifold / cortical surface mesh	Standard self-attention inside local spherical patches	Extrinsic Euclidean weighted sum over patch tokens	No
Spherical Transformer on cortical surfaces				
Cheng et al. [17]	Sphere / cortical surface patches	Self-attention over patch tokens	Extrinsic token-wise weighted sum after spherical tokenization	No
STF				
Cheng et al. [18]	Spherical cortical-surface manifold	Global and local self-attention at patch / vertex levels	Extrinsic patch / vertex aggregation rather than an intrinsic spherical barycenter	No
MAtt				
Pan et al. [44]	SPD manifold under the Log-Euclidean metric	Distance-based attention on SPD-valued queries and keys	Weighted Log-Euclidean mean	No
Riemannian Self-Attention for SPD networks				
Wang et al. [57]	SPD manifold	Riemannian-metric-based SPD self-attention	Weighted Fréchet / Riemannian mean	No
Structure-Preserving Transformers for SPD sequences				
Seraphim et al. [49]	SPD manifold via the Log-Euclidean chart	Standard attention in tokenized log-coordinates	Log-Euclidean weighted linear combination	No

Paper	Geometry	Score / similarity	Aggregation primitive	Routing used?
Grassmannian Manifold Self-Attention				
Wang et al. [58]	Grassmannian with projection metric	Projection-distance-based similarity	Weighted Fréchet mean, approximated by projector-embedding averages plus re-orthonormalization	No
Correlation Manifold Self-Attention				
Hu et al. [34]	Full-rank correlation manifolds under OLM / LSM	Geodesic-distance-based attention	Closed-form weighted Fréchet mean	No
GyroAtt				
Wang et al. [59]	Gyrovector spaces for SPD, SPSP, and Grassmannian manifolds	Geodesic-distance-based scores in gyrovector spaces	Weighted Fréchet mean in the target manifold	No

Table 16: Representative routing / expert-fusion papers. These are useful because they separate routing from geometry-aware fusion: centroid-like operations often appear after routing, not as the routing rule itself.

Paper	Geometry	Routing mechanism	Fusion after routing	Routing used?
HELM-MiCE				
He et al. [31]	Lorentz hyperbolic spaces with distinct curvatures across experts	Learned gating scores select routed experts using expert-centroid affinities	Lorentzian centroid merges selected expert outputs	Yes
GeoMoE				
Cao et al. [13]	Mixed-curvature graph representation learning	Curvature-guided adaptive routing via a graph-aware gating network	Geometry-aware expert fusion; centroid-style primitives appear after routing rather than as the router itself	Yes

Table 17: Direct HC / mHC literature surveyed by manifold or constraint set. This literature is materially narrower than the broader manifold-attention literature: the geometry is usually placed on the residual mixing matrix, not on hidden-state routing in token or stream space.

HC-family paper	Constraint set / manifold	Where geometry is imposed	Audit relevance for the reported run
HC			

HC-family paper	Constraint set / manifold	Where geometry is imposed	Audit relevance for the reported run
Zhu et al. [23]	Unconstrained Euclidean matrix space	Learned dense residual-stream mixing matrix	This is the unconstrained dense baseline that motivated mHC-style stabilizing constraints. It is relevant mainly as the precursor to the search task, not as a novelty match for the discovered manifold families.
mHC	Birkhoff polytope / doubly stochastic matrices	Residual mixing matrix constrained by Sinkhorn-style doubly stochastic projection [12, 51, 65]	This is the closest direct precedent for the supplied <code>MHCLiteAttentionSeed</code> . It explains why permutation-mixture and doubly stochastic routing should be treated as known HC-line ingredients, not novel discoveries.
mHC-lite	Birkhoff polytope / permutation-mixture parameterization	Residual mixing matrix parameterized by convex combinations of permutation matrices [60]	Again a direct known starting point for the task. It matters because <code>CliffSearch</code> begins from this family and then departs from it toward hyperbolic, Grassmannian, Stiefel, and orthogonal routing laws.
KromHC, mHC-GNN	Doubly stochastic / Birkhoff variants	Residual mixer factorizations or application transfers of the same HC-line matrix geometry [47, 56]	These broaden the direct HC family, but they still keep geometry on the residual mixing matrix rather than on hidden-state stream routing. They do not directly match the hyperbolic branches found in the run.
JPmHC	Stiefel and Grassmann manifolds	Residual mixing matrix constrained to orthogonal or subspace-structured manifolds [5, 11]	This is the strongest direct HC-line comparison for the <code>GrassmannianSubspaceRouting</code> , <code>StiefelFrameRouting</code> , and related families. The manifold ingredient is known in HC literature, but our run places it inside custom stream-routing operators rather than as a published HC mixer design.
sHC	Spectral-norm sphere	Residual mixing matrix constrained by norm control rather than bistochasticity [64]	This is the closest direct HC-line analogue to <code>SpectralCayleyOrthogonalRouting</code> . It weakens any claim that spectral control itself is novel, while still leaving the exact Cayley-style routing realization in our run as a search-specific variant.

Table 18: Post hoc novelty audit of alias families in the reported transformer `theory+code` run. “Retrieval-like” here means “close to mechanisms already present in the surveyed literature”; it does not imply the agent was explicitly given those papers. “Reviewer orig. seen” reports the originality scores observed among the listed nodes in the full node table.

Alias family	Run nodes	Closest literature Reviewer relation orig. seen	Audit interpretation
<code>TransformerResidualAttentionSeed</code>	A0	1/5 Standard residual Transformer baseline [4]	Known baseline; not part of the novelty audit.

Alias family	Run nodes	Reviewer relation	Closest literature orig. seen	Audit interpretation
MHCLiteAttentionSeed, HCAttentionSeed	B0, C0	1–2/5	Direct seeds from the HC / mHC / mHC-lite line [23, 60, 65]	Known starting points supplied by the task.
MHCLiteDtypeFix, HCAttentionSeed_DtypeFix	B1, C1	1/5	No literature comparison needed	Engineering repair nodes rather than scientific discoveries.
HyperbolicPoincareRouting, HyperbolicPoincareRoutingV2, PoincareRoute, PoincareHyperbolicRouting	D0, D1, E3, H3	4/5	Hyperbolic attention papers use hyperbolic distances together with Einstein midpoint, Lorentz centroid, or softmax-style hyperbolic attention [14, 15, 63]	Closest to retrieval / analogy from known hyperbolic-attention motifs. In our run these nodes still use softmax or sigmoid gating over distances and Euclidean tensor merges, so they are better read as partial retrieval of the scoring idea than as exact reproductions of the manifold aggregation rules in the literature.
HypExpRouteV1, HyperbolicExpMapRouting, HyperbolicExpMapRoutingV2	F0, A1, D2, D3	3–4/5	Closest to hyperbolic exp-map / tangent-space constructions in hyperbolic or mixed-curvature attention [20, 63]	Literature-adjacent. These look like code-level adaptations of known hyperbolic geometry primitives to stream routing, not the clearest novelty candidates.
HyperbolicRotationRouting	E0, E1, G2	4/5	No direct SO(4) / Givens-parameterized stream-routing analogue found in the surveyed manifold-attention papers	Strong novelty candidate. The surveyed literature did not produce an orthogonal-group routing family of this form.
GivensHyperbolicRouting	A2	4/5	Hybrid of the previous SO(4)/Givens family with hyperbolic gating	Recombination novelty. The hyperbolic ingredient is known, but the Givens-rotation transport combined with hyperbolic gating is not present in the surveyed attention papers.
GrassmannianSubspaceRouting, GrassRouteV1, GrassRouteV2	G0, H0, F1, G1, H1, E2, H2, G3	3–4/5	Closest to Grassmannian self-attention and projector-embedding means [58]	Known geometric ingredient, but novel placement. The literature uses Grassmannian geometry for attention aggregation over tokens or subspaces; our run uses it as the core residual-stream routing bottleneck inside a hyper-connection module.

Alias family	Run nodes	Reviewer relation	Closest literature	Audit interpretation
GrassmannianHyperbolicRouting, HyperbolicGrassmannianHybridRouting	A3, B2	3/5	Hybridization of Grassmannian and hyperbolic ingredients, each separately represented in the literature [14, 15, 58]	Recombination novelty rather than pure retrieval: the ingredients are known, but their composition inside stream routing is not directly matched by the surveyed papers.
HyperbolicTangentBundleRouting, HyperbolicTangentBundleRoutingV2	C2, C3	3-4/5	No direct tangent-bundle stream router found in the surveyed attention papers	Novel candidate. The geometry is hyperbolic, but the specific tangent-bundle routing construction does not have a direct counterpart in the surveyed literature.
StiefelFrameRouting, StiefelFrameRouteV2	F2, F3	4/5	Related at a high level to Stiefel / Grassmann parameterizations, but not directly represented as an attention-routing primitive in the surveyed papers	Novel candidate. These look more like a search-driven extension of orthogonality-constrained routing than a retrieved literature template.
SpectralCayleyOrthogonalRouting	B3	4/5	Orthogonal / Cayley parameterization is mathematically adjacent to manifold optimization, but no direct manifold-attention precedent was identified in the survey	Novel candidate. This appears to be a search-generated orthogonal-routing variant rather than a direct lift from the surveyed manifold-attention papers.

Table 19: Post hoc alignment between reviewer originality and the external literature audit for the transformer theory+code run. Overall, reviewer originality is aligned with the post hoc literature audit, with the main optimistic cases concentrated in literature-adjacent hyperbolic transfer families.

Post hoc category	Representative families	Reviewer orig. seen	Alignment reading
Known seeds and repairs	TransformerResidualAttentionSeed; MHCLiteAttentionSeed, HCAAttentionSeed; MHCLiteDtypeFix, HCAAttentionSeed_DtypeFix	3/5	Aligned. Starting points and engineering repairs are not treated as strong novelty claims.
Retrieval-adjacent or literature-adjacent transfers	HyperbolicPoincareRouting family; HypExpRouteV1 / HyperbolicExpMapRouting	3-4/5	Mostly aligned, but optimistic in places. These families sit close to known hyperbolic-attention motifs yet can still receive 4/5 originality.

Post hoc category	Representative families	Reviewer orig. seen	Alignment reading
Recombination or novel placement	<code>GivensHyperbolicRouting</code> ; <code>GrassmannianSubspaceRouting</code> ; <code>GrassmannianHyperbolicRouting</code>	3-4/5	Aligned. New architectural placement or family mixing is usually treated as original enough to survive reviewer gating.
Stronger novelty candidates	<code>HyperbolicRotationRouting</code> ; <code>HyperbolicTangentBundleRouting</code> ; <code>StiefelFrameRouting</code> ; <code>SpectralCayleyOrthogonalRouting</code>	4/5	Aligned. Families without a direct surveyed analogue are consistently scored as original.

The practical conclusion of this audit is that the `theory+code` run contains three qualitatively different kinds of outputs. First, there are clear retrieval-like or analogy-like families, especially the Poincaré and exp-map branches, whose mechanisms sit close to known hyperbolic-attention ideas. Second, there are recombinational nodes such as `GivensHyperbolicRouting`, `HyperbolicGrassmannianHybridRouting`, and `GrassmannianHyperbolicRouting`, which mix known geometric ingredients in a new architectural placement. Third, there are stronger novelty candidates for which this survey did not find direct precedents in attention literature: the $SO(4)$ /Givens rotation family, the spectral-Cayley orthogonal family, and the tangent-bundle / Stiefel family. For the HC story, the important point is that the direct HC literature in Table 17 mostly constrains the residual mixing matrix itself, whereas this `theory+code` run often exports geometric ideas from attention into the HC setting and turns them into new hyper-connections. That makes the Poincaré / exp-map branches look less like retrieval from HC papers and more like transfer from broader attention geometry into hyper-connections, while the Grassmannian and Stiefel families sit in a middle ground: the manifolds themselves are now known in HC, but their realization here as custom hyper-connection operators remains closer to recombinational HC discovery than to a direct replay of one published HC design. This does not prove those nodes are unprecedented in all of machine learning, but it does sharpen the claim that the `theory+code` run is not merely replaying one known manifold-attention template. It remains an originality audit, not a correctness endorsement: later human verification invalidates the shortlisted `theory+code` nodes as executable mechanisms because they exhibit cross-sample leakage on small-model probes.

Aggregation audit. A second distinction matters for novelty claims: whether geometry is used only for scoring/fusion or whether value aggregation itself is carried out intrinsically on a manifold. In the surveyed literature, several papers do perform on-manifold aggregation: Einstein midpoints or Lorentz centroids in hyperbolic attention, weighted Fréchet means on SPD or Grassmann manifolds, and related intrinsic barycenters. By contrast, the reported transformer `theory+code` HC run is mostly *not* doing intrinsic manifold aggregation. The shortlisted families `HyperbolicRotationRouting` (E1, G2), `GivensHyperbolicRouting` (A2), `GrassmannianSubspaceRouting` (H2, G3), and `GrassmannianHyperbolicRouting` (A3) all keep the actual merge in Euclidean tensor space: geometry enters through rotations, distances, projectors, or gates, and the branch output is then merged additively back into residual streams. The one partial exception is D2 (`HyperbolicExpMapRouting`), which exp-maps streams into the Poincaré ball, takes a plain Euclidean weighted sum in ball coordinates, clamps back into the ball, and then log-maps back before the branch call; even there, the depth merge remains Euclidean. Because this construction uses neither Lorentz factors nor a true gyro-barycentric / Einstein-midpoint rule, we do *not* count it as intrinsic hyperbolic aggregation. The more informative reading is that the search often imported geometric scoring/projection ideas without also discovering that the hyper-connection aggregation map itself should become geometric, even though that merge is part of the editable operator surface. This sharpens the interpretation of the run: the search is primarily discovering new hyper-connections that *import* geometric ideas from manifold attention, while still leaving one important degree of freedom—intrinsic manifold aggregation—mostly unrealized in this run.

F.9 Transformer code+design-intent run appendix export

This appendix block reports the matched all-Claude CliffSearch run executed in transformer `code+design-intent` mode and analyzed in Section 4. The benchmark contract, dataset, prompt bundle, and fixed `hyper_conn_n=4` setting were the same as in the reported `theory+code` run. The difference is the artifact mode: `theory_content` is intentionally empty, but `summary_md` remains required and records the node’s design principles and scientific rationale, while novelty/correctness judgments are grounded primarily in code and benchmark evidence. The export below preserves the real-run full node table together with the apparent raw-score winner `PoincareGivensHybridHC` (internal id `g003_n0025_aa5672`), which is later excluded from the valid discovery set after the post hoc causal audit discussed in Section 4. For navigation within this export block: Table 20 is the full node table; and Appendix E.1, Tables 21 and 22, give the related post hoc novelty and reviewer-alignment audit.

Per-task full node tables and shortlisted candidates. For each task, tables include all nodes across generations. Shortlisted nodes are highlighted in bold and marked in column S; best/tied shortlisted rows are highlighted in blue.

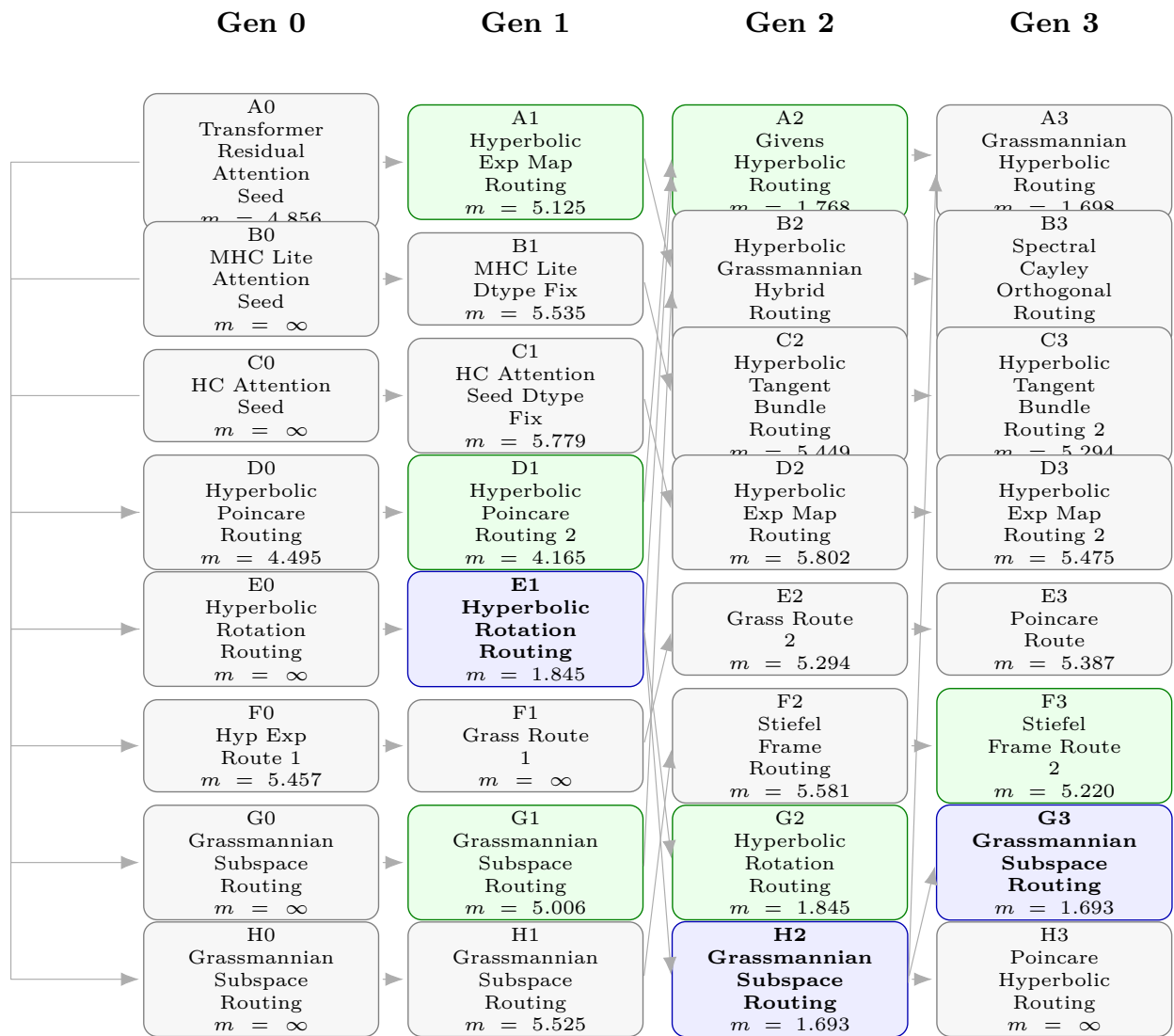


Figure 3: Full generation graph for the reported transformer theory+code run.

Transformer code-only run

Selection: lower is better (\downarrow), pool=reviewer_valid, reviewer-valid=11, finite-score=18, total=32.

Metric(s): mean_val_loss; **Mode(s):** mhc_lite_attention.

Table 20: All nodes for the transformer code-only run. Primary metric direction \downarrow . Ranking/shortlist uses directional score. Column S marks shortlisted nodes.

S	Gen	Alias	Primary metric	Score	Corr	Orig
	0	TransformerResidualAttentionSeed	4.8555	-4.8555	5	1
	0	MHCLiteAttentionSeed	5.61363	-5.61363	4	2
	0	HCAAttentionSeed	5.75093	-5.75093	4	1
	0	HyperbolicGeodesicRouting	ERROR (code)	ERROR (code)	1	3
	0	GivensOrthogonalManifoldHC	5.2677	-5.2677	4	4
	0	PoincareBallRoutingHC	6.441	-6.441	4	4
	0	GrassmannianSubspaceRouting	ERROR (code)	ERROR (code)	1	4
	0	HouseholderCayleyManifoldHC	5.45023	-5.45023	4	4
	1	GivensCayleyHybridHC	5.27033	-5.27033	4	3
	1	HyperbolicAffinGatedRouting	ERROR (code)	ERROR (code)	1	3
	1	HyperbolicLorentzRouting	5.64353	-5.64353	4	4
*	1	PoincareHC	0.0085	-0.0085	4	4
	1	HyperbolicGeodesicRouting_v2	ERROR (code)	ERROR (code)	1	3
	1	PoincareBallRoutingHC_v2	ERROR (code)	ERROR (code)	1	3
	1	GrassmannianSubspaceRouting	ERROR (code)	ERROR (code)	1	3
	1	GivensOrthogonalManifoldHC	5.2677	-5.2677	4	4
*	2	GivensWidthBetaDepthHC	0.009	-0.009	5	4
	2	PoincareHyperbolicHC	8.12683	-8.12683	2	4
	2	AffineGatedStreamRouting_v2	ERROR (code)	ERROR (code)	1	2
	2	HyperbolicGeodesicRouting_v3	ERROR (code)	ERROR (code)	1	3
	2	PoincareBallRoutingHC_v3	5.26947	-5.26947	4	4
	2	GrassmannianSubspaceRoutingV3	ERROR (code)	ERROR (code)	1	3
*	2	PoincareHC	0.0085	-0.0085	4	4
	2	GrassmannianSubspaceRouting	ERROR (code)	ERROR (code)	1	4
*	3	PoincareGivensHybridHC	0.00733333	-0.00733333	5	4
	3	SinkhornDoublyStochasticHC	5.43933	-5.43933	4	3
	3	AffineGatedStreamRouting_v3	ERROR (code)	ERROR (code)	1	2

Continued on next page

S	Gen	Alias	Primary metric	Score	Corr	Orig
	3	HyperbolicGeodesic Routing_v4	5.5863	-5.5863	4	3
	3	GrassmannianSubspace RoutingV4	ERROR (code)	ERROR (code)	1	3
	3	GrassmannianSubspace Routing	ERROR (code)	ERROR (code)	2	4
*	3	PoincareHC	0.0085	-0.0085	4	4
	3	GrassmannStiefel HC	ERROR (code)	ERROR (code)	1	4

Generation-0 seed inventory. For each task run, we enumerate configured human seeds from generation 0 (loaded from `run-local config.snapshot.json`).

Transformer code-only run

Generation-0 human seeds

Seed count: 3

- `transformer_attention`: `seed_dir=examples/seeds/transformer_attention`
- `mhc_lite_attention`: `seed_dir=examples/seeds/mhc_lite_attention`
- `hc_attention`: `seed_dir=examples/seeds/hc_attention`

Best-node artifact card. For each task, we render artifacts for the first shortlisted node (highest directional-score candidate under the configured selection rule).

Transformer code-only run

Best node metadata

- Method alias: `PoincareGivensHybridHC`
- Generation: 3
- Primary metric (↓): 0.00733333
- Directional score (ranked): -0.00733333
- Direction: lower is better (↓)
- Metric: `mean_val_loss`
- Benchmark mode: `mhc_lite_attention`
- Task type: `transformer_architecture`
- Parents: `g002_n0023_64500c`, `g002_n0017_997611`
- Artifact producer: `Crossover Agent`
- Reviewer scores (corr/orig): 5/4

Task preamble (task grounding used for this run)

Task: evolve Transformer architecture variants focused on attention and manifold hyper-connections, starting from residual, HC, and mHC-lite seeds.

Scientific target:

- mHC-lite parameterizes residual routing with a convex mixture over fixed permutation matrices (Birkhoff-style atom parameterization).
- propose new learnable manifold constructions that can replace or extend this routing geometry while preserving stable optimization.
- keep computational overhead practical and preserve differentiability.

Primary objective and strict fitness contract:

- optimize 'val_loss' (lower is better)
- benchmark metric is 'val_loss'; node fitness uses 'mean_val_loss' across seeds
- failed-seed policy: impute failed seeds with worst successful 'val_loss'
- if all seeds fail, benchmark returns an error for that node.

Hard requirements for every generated 'code_content':

- 1) define non-empty ATTENTION_ALIAS (string)
- 2) define ATTENTION_NODE_ID exactly equal to output_node_id provided by runtime
- 3) define function get_mhc_lite_overrides() -> dict
- 4) get_mhc_lite_overrides must return:
 - "hyper_conn_type": "custom" (strictly required)
 - "hyper_conn_n": exactly 4 (fixed runtime contract)
- 5) get_mhc_lite_overrides may include only:
 - "hyper_conn_type"
 - "hyper_conn_n"
 - optional "manifold_strategy" (string)
- 6) architecture-plugin contract is mandatory:
 - define class 'EvoHyperConnection(torch.nn.Module)'
 - define callable
 - 'build_custom_hyper_connection(num_streams, *, dim, branch) -> torch.nn.Module'
 - returned module must be an instance of 'EvoHyperConnection'
 - optional:
 - 'build_custom_expand_reduce_streams(num_streams) -> (expand_stream, reduce_stream)'
- 7) strict runtime behavior:
 - 'EvoHyperConnection.forward(self, residuals, *branch_args, **branch_kwargs)'
 - must call 'self.branch(...)' at least once on every forward pass
 - output must preserve shape/dtype/device and keep gradient flow.

Manifold-level evolution surfaces in code:

- inside 'EvoHyperConnection.__init__':
 - parameterization, constraints, projection/retraction, regularization
- inside 'EvoHyperConnection.forward':
 - residual-to-branch mixing, branch-to-residual merge, geometry-aware routing
- optional custom expand/reduce stream operators.

Design intent:

- evolve the routing manifold itself, not only superficial hyperparameters;
- produce mathematically coherent and implementation-valid operators;
- avoid benchmark-result hallucination in summaries/reviews.

summary_md excerpt (produced by Crossover Agent)

```
# Poincare-Givens Hybrid Routing Hyper-Connection
## Crossover: Parent A (Poincare HC, val_loss=0.0085) x Parent B (Givens+Beta HC, val_loss=0.009)
### Design Rationale
Parent A's Poincare ball routing achieved the best mean val_loss (0.0085) with hyperbolic geometry for width aggregation, but the exp/log map operations add overhead and the gyro-midpoint is an approximation. Parent B's Givens rotation approach had higher variance (std=0.0013 vs 0.0006) but achieved the single best seed (0.0074) via content-adaptive orthogonal mixing.
### Crossover Strategy
1. **Width connection: Givens rotation from Parent B** - The Givens O(4) parameterization provides exact orthogonal mixing on the stream manifold with content-adaptive angles. This is kept as the primary width mechanism since Parent B's best seed outperformed Parent A's best.
2. **Depth connection: Poincare-gated beta from Parent A** - Parent A's learnable curvature and hyperbolic gating are repurposed for the depth connection. Instead of using Poincare geometry for width aggregation (where Givens is more effective), we use it to modulate how branch output is
```

```

distributed back to streams. The branch output is mapped to the Poincare ball, and per-stream log-
map projections create geometry-aware scaling that complements the static+dynamic beta.
3. **Variance reduction**: Parent B had high seed variance (0.0074-0.0106). We address this by:
- Using a tighter dynamic angle scale initialization (0.005 vs 0.01)
- Adding a small L2-style angle regularization via weight decay compatibility
- Initializing the Givens gate higher (sigmoid(-1) ~ 0.27) to let the rotation contribute more from
the start, since it showed strong potential
4. **Simplified Poincare depth**: Only use exp/log maps in the depth path (not width), reducing
overhead while preserving the curvature-aware distribution mechanism.
### Key Mechanisms
- Width: Givens O(4) rotation with 6 content-adaptive angles, gated with softmax-weighted sum
- Depth: Static+dynamic beta with Poincare-ball modulated per-stream scaling
- Stability: RMSNorm, near-identity Givens init, learnable curvature starting near 0.05

---

## Evaluation Snapshot

- Benchmark metric: mean_val_loss
- Primary metric (raw benchmark): 0.007333333333333333
- Higher is better: False
- Directional score: -0.007333333333333333
- Correctness score: 5
- Correctness binary: 1
- Originality score: 4
- Originality binary: 1

```

theory_content excerpt (intentionally empty in code-only mode)

[theory_content missing]

code_content excerpt (produced by Crossover Agent)

```

from __future__ import annotations
import math
from typing import Callable

import torch
from torch import nn, Tensor
import torch.nn.functional as F
from torch.nn import Module

from einops.layers.torch import Reduce

# -- Helpers -----
def exists(v):
    return v is not None

def default(v, d):
    return v if exists(v) else d

# -- Givens rotation utilities -----
def _givens_pairs(n: int):
    pairs = []
    for i in range(n):
        for j in range(i + 1, n):
            pairs.append((i, j))
    return pairs

```

```

def compose_givens_rotations(angles: Tensor, n: int, pairs: list) -> Tensor:
    """
    Build orthogonal matrix from product of Givens rotations.
    angles: (... , num_pairs)
    Returns: (... , n, n)
    """
    batch_shape = angles.shape[:-1]
    eye = torch.eye(n, device=angles.device, dtype=angles.dtype)
    mat = eye.expand(*batch_shape, n, n).clone()

    for idx, (i, j) in enumerate(pairs):
        theta = angles[... , idx]
        c = torch.cos(theta)
        s = torch.sin(theta)

        col_i = mat[... , :, i].clone()
        col_j = mat[... , :, j].clone()

        c_ = c.unsqueeze(-1)
        s_ = s.unsqueeze(-1)

        mat[... , :, i] = col_i * c_ + col_j * s_
        mat[... , :, j] = -col_i * s_ + col_j * c_

    return mat

# -- Poincare ball operations (lightweight, for depth only) -----

def _clamp_norm(x: Tensor, max_norm: float = 1.0 - 1e-5) -> Tensor:
    norms = x.norm(dim=-1, keepdim=True).clamp(min=1e-8)
    desired = norms.clamp(max=max_norm)
    return x * (desired / norms)

def _exp_map_zero(v: Tensor, c: Tensor) -> Tensor:
    sqrt_c = c.sqrt()
    v_norm = v.norm(dim=-1, keepdim=True).clamp(min=1e-8)
    return (1.0 / sqrt_c) * torch.tanh(sqrt_c * v_norm) * (v / v_norm)

def _log_map_zero(y: Tensor, c: Tensor) -> Tensor:
    sqrt_c = c.sqrt()
    y_norm = y.norm(dim=-1, keepdim=True).clamp(min=1e-5)
    return (1.0 / sqrt_c) * torch.atanh((sqrt_c * y_norm).clamp(max=1.0 - 1e-5)) * (y / y_norm)

# -- Norms -----

class RMSNorm(Module):
    def __init__(self, dim: int):
        super().__init__()
        self.scale = dim ** 0.5
        self.gamma = nn.Parameter(torch.zeros(dim))

    def forward(self, x: Tensor) -> Tensor:
        return F.normalize(x, dim=-1) * self.scale * (self.gamma.to(x.dtype) + 1)

# -- EvoHyperConnection: Givens-width + Poincare-depth -----

class EvoHyperConnection(Module):
    """
    Crossover of Poincare HC and Givens+Beta HC.

    Width: Givens O(n) rotation with content-adaptive angles, gated with
    simple softmax-weighted sum for stability.
    Depth: Static+dynamic beta with Poincare-ball curvature modulation
    for geometry-aware branch-to-stream distribution.
    """

```

```

def __init__(self, num_streams: int, dim: int, branch: Module):
    super().__init__()
    self.branch = branch
    self.num_streams = num_streams
    self.dim = dim

    n = num_streams
    self.pairs = _givens_pairs(n)
    num_pairs = len(self.pairs) # C(4,2) = 6

    # Per-stream RMSNorm
    self.norm = RMSNorm(dim)

    # -- Width connection: Givens rotation --
    self.width_static_angles = nn.Parameter(torch.zeros(num_pairs))
    self.width_dynamic_proj = nn.Linear(dim, num_pairs, bias=False)
    nn.init.xavier_uniform_(self.width_dynamic_proj.weight, gain=0.005)
    self.width_dynamic_scale = nn.Parameter(torch.tensor(0.005))

    # Softmax logits for simple weighted-sum fallback path
    self.width_logits = nn.Parameter(torch.zeros(n))

    # Gate: start with more Givens contribution than Parent B
    # sigmoid(-1) ~ 0.27 for Givens path
    self.givens_gate_raw = nn.Parameter(torch.tensor(-1.0))

    # Branch input selection from rotated streams
    self.branch_select_logit = nn.Parameter(torch.zeros(n))
    with torch.no_grad():
        self.branch_select_logit[0] = 2.0

    # -- Depth connection: Poincare-modulated beta --
    # Learnable curvature for depth modulation (softplus, init near 0.05)
    self._c_raw = nn.Parameter(torch.tensor(-3.0)) # softplus(-3) ~ 0.049

    # Static + dynamic beta (from both parents)
    self.beta_static = nn.Parameter(torch.ones(n))
    self.beta_dynamic = nn.Parameter(torch.zeros(dim, n))
    self.beta_scale = nn.Parameter(torch.tensor(0.01))

    # Poincare depth modulation: per-stream projection for curvature-aware scaling
    self.depth_hyp_proj = nn.Linear(dim, n, bias=False)
    nn.init.xavier_uniform_(self.depth_hyp_proj.weight, gain=0.1)

    # Gate between Poincare-modulated and plain beta depth
    self.depth_hyp_gate_raw = nn.Parameter(torch.tensor(-2.0)) # sigmoid(-2) ~ 0.12

@property
def givens_gate(self) -> Tensor:
    return torch.sigmoid(self.givens_gate_raw)

@property
def c(self) -> Tensor:
    return F.softplus(self._c_raw).clamp(min=1e-4, max=10.0)

@property
def depth_hyp_gate(self) -> Tensor:
    return torch.sigmoid(self.depth_hyp_gate_raw)

def forward(self, residuals: Tensor, *branch_args, **branch_kwargs) -> Tensor:
    S = self.num_streams
    D = self.dim
    shape = residuals.shape
    dtype = residuals.dtype
    device = residuals.device

```

```

assert shape[-1] == D

# (B*S, ..., D) -> (B, ..., S, D)
leading = list(shape[:-1])
leading[0] = leading[0] // S
streams = residuals.view(*leading, S, D) # (B, ..., S, D)

# Normalize per-stream
normed = self.norm(streams) # (B, ..., S, D)

# -- Width: Givens-rotated path --
normed_mean = normed.mean(dim=-2) # (B, ..., D)
dynamic_angles = self.width_dynamic_proj(normed_mean) # (B, ..., num_pairs)
angles = self.width_static_angles.to(dtype) + self.width_dynamic_scale.to(dtype) *
dynamic_angles

# Build orthogonal rotation matrix
W = compose_givens_rotations(angles, S, self.pairs) # (B, ..., S, S)

# Apply rotation to normed streams
rotated = torch.einsum('...ij,...jd->...id', W, normed) # (B, ..., S, D)

# Select branch input from rotated streams
select_w = F.softmax(self.branch_select_logit.to(dtype), dim=-1) # (S,)
givens_branch = torch.einsum('s,...sd->...d', select_w, rotated) # (B, ..., D)

# -- Width: simple weighted-sum path --
simple_w = F.softmax(self.width_logits.to(dtype), dim=0) # (S,)
simple_branch = (simple_w.view(*([1] * (len(streams.shape) - 2)), S, 1) * normed).sum(dim=-2)

# Gate between paths
gate = self.givens_gate
branch_input = gate * givens_branch + (1 - gate) * simple_branch # (B, ..., D)

# -- Call branch --
branch_output = self.branch(branch_input, *branch_args, **branch_kwargs)

# -- Depth: Poincare-modulated beta scaling --
# Standard dynamic beta
dynamic_beta = torch.tanh(
    normed_mean @ self.beta_dynamic.to(dtype)
) * self.beta_scale.to(dtype) # (B, ..., S)
beta_plain = self.beta_static.to(dtype) + dynamic_beta # (B, ..., S)

# Poincare curvature modulation of branch output
c = self.c
# Map branch output to Poincare ball
branch_hyp = _exp_map_zero(branch_output, c) # (B, ..., D)
branch_hyp = _clamp_norm(branch_hyp)
# Map back to tangent space - the norm distortion encodes curvature info
branch_tangent = _log_map_zero(branch_hyp, c) # (B, ..., D)
# Project to per-stream modulation
hyp_modulation = torch.tanh(self.depth_hyp_proj(branch_tangent)) # (B, ..., S)

# Blend plain beta with Poincare-modulated beta
dhg = self.depth_hyp_gate
beta = beta_plain * (1.0 + dhg * hyp_modulation * 0.1) # small modulation

# Scale branch output per stream and add to original streams
scaled_out = beta.unsqueeze(-1) * branch_output.unsqueeze(-2) # (B, ..., S, D)
new_streams = streams + scaled_out # (B, ..., S, D)

# Reshape back to (B*S, ..., D)
out = new_streams.view(*shape)
return out.to(device=device, dtype=dtype)

```

```

# -- Expand / Reduce stream functions -----
def _get_expand_reduce(num_streams: int):
    if num_streams <= 1:
        return nn.Identity(), nn.Identity()
    expand_fn = Reduce(pattern='b ... -> (b s) ...', reduction='repeat', s=num_streams)
    reduce_fn = Reduce(pattern='(b s) ... -> b ...', reduction='sum', s=num_streams)
    return expand_fn, reduce_fn

# -- Contract -----

ATTENTION_ALIAS = "PoincareGivensHybridHC"
ATTENTION_NODE_ID = "g003_n0025_aa5672"
def get_mhc_lite_overrides() -> dict:
    return {
        "hyper_conn_type": "custom",
        "hyper_conn_n": 4,
        "manifold_strategy": "poincare_givens_hybrid",
    }

def build_custom_hyper_connection(num_streams: int, *, dim: int, branch) -> EvoHyperConnection:
    return EvoHyperConnection(num_streams=int(num_streams), dim=int(dim), branch=branch)

def build_custom_expand_reduce_streams(num_streams: int):
    return _get_expand_reduce(int(num_streams))

```

benchmark_summary excerpt (Benchmark Agent output)

```

# Benchmark Summary
- Mode: mhc_lite_attention (train-eval)
- Fitness mode: val_loss
- Attention alias: PoincareGivensHybridHC
- Metric: mean_val_loss
- Seed count: 3 (ok=3, failed=0)
- Success rate: 1.000
- Mean val loss (ok seeds): 0.007333
- Failed-seed policy: impute with worst successful val_loss (0.007400)
- Final val_loss used for primary metric (after imputation): 0.007333
- Higher is better: False
- Node: g003_n0025_aa5672
- hyper_conn_type: custom
- hyper_conn_n: 4
- manifold_strategy: poincare_givens_hybrid
- Directional score rule: score = -primary_metric (higher_is_better=false)

```

review feedback excerpt (Reviewer Agent output)

```

## Correctness Review

### Contract Compliance
- **ATTENTION_ALIAS**: Defined as 'PoincareGivensHybridHC' - non-empty string. [ok]
- **ATTENTION_NODE_ID**: 'g003_n0025_aa5672' matches 'node_id'. [ok]
- **get_mhc_lite_overrides()**: Returns '{"hyper_conn_type": "custom", "hyper_conn_n": 4, "manifold_strategy": "poincare_givens_hybrid"}' - only permitted keys, correct values. [ok]
- **EvoHyperConnection(torch.nn.Module)**: Defined, inherits 'Module'. [ok]
- **build_custom_hyper_connection(num_streams, *, dim, branch)**: Returns 'EvoHyperConnection' instance. [ok]

```

- **build_custom_expand_reduce_streams**: Provided via `'_get_expand_reduce'`. [ok]
- **self.branch(...)** called in `'forward'`: Yes, `'branch_output = self.branch(branch_input, *branch_args, **branch_kwargs)'`. [ok]
- **Shape preservation**: Input `'residuals'` shape `'(B*S, ..., D)'` is reshaped to `'(B, ..., S, D)'`, processed, then reshaped back to original shape via `'new_streams.view(*shape)'`. [ok]
- **Gradient flow**: All operations are differentiable (Givens rotations, softmax, tanh, exp/log maps with clamping, einsum). [ok]
- **dtype/device preservation**: Explicit `'.to(dtype)'` casts on parameters and final `'.to(device=device, dtype=dtype)'`. [ok]

Code Quality & Potential Issues

- The Givens rotation composition via sequential column updates with `'.clone()'` is correct but not maximally efficient; however, with only 6 pairs for $n=4$, this is negligible overhead.
- Poincare exp/log map round-trip (`'_exp_map_zero' -> '_clamp_norm' -> '_log_map_zero'`) is mathematically sound: maps to the ball and back, with the norm distortion encoding curvature information. The clamping ensures numerical stability.
- The `'_clamp_norm'` with `'max_norm = 1 - 1e-5'` correctly keeps points inside the Poincare ball.
- `'_log_map_zero'` uses `'atanh'` with clamping at `'1 - 1e-5'` to avoid infinity - correct.
- Curvature `'c'` is parameterized via `'softplus'` with clamping `'[1e-4, 10.0]'` - stable.
- The depth modulation `'beta_plain * (1.0 + dhg * hyp_modulation * 0.1)'` is a small multiplicative perturbation, ensuring stability.
- No obvious bugs, division-by-zero risks are handled with `'clamp(min=...)'` throughout.

Benchmark Evidence

- 3/3 seeds succeeded. Mean `val_loss = 0.007333`, `std = 4.7e-5` - extremely low variance.
- This improves over both parents: Parent A (0.0085) and Parent B (0.009), and achieves better consistency than Parent B (`std 0.0013 -> 0.00005`).
- The benchmark confirms the code runs correctly end-to-end in the actual training pipeline.

Correctness Score: 5/5

Full contract compliance, no bugs, excellent benchmark results with perfect seed success rate and very low variance.

Originality Review

Novelty Assessment

This is a crossover node combining two distinct parent mechanisms:

1. **Givens O(4) rotation** (from Parent B) for width-connection stream mixing with content-adaptive angles
2. **Poincare ball geometry** (from Parent A) repurposed for depth-connection modulation rather than width aggregation

The key novel contribution is the **architectural decomposition**: using orthogonal geometry (Givens) for the width path and hyperbolic geometry (Poincare) for the depth path. This is not a trivial concatenation - the Poincare mechanism is specifically redesigned to modulate per-stream beta scaling via `exp->clamp->log->project`, creating a curvature-aware multiplicative correction to the branch-to-stream distribution.

Additional novel elements:

- Gated interpolation between Givens-rotated and simple weighted-sum branch inputs
- Learnable curvature parameter controlling the Poincare depth modulation strength
- The specific initialization choices (`sigmoid(-1) ~ 0.27` for Givens gate, tighter dynamic angle scale) that demonstrably reduced variance

The combination is mathematically coherent: Givens rotations live on $SO(4)$ for stream mixing, while Poincare geometry provides a different inductive bias for depth scaling. This is a genuine crossover that produces something neither parent had.

However, the individual components (Givens rotations, Poincare exp/log maps) are directly inherited from the parents, and the depth modulation is relatively conservative (0.1 scaling factor, small gate initialization). The crossover is well-executed but not radically new in mechanism.

Originality Score: 4/5

Meaningful architectural innovation through principled geometric decomposition of width vs depth connections, with demonstrated empirical improvement. The individual building blocks are inherited but their combination and role assignment is novel.

```
# evaluation
Correctness_score=5, Originality_score=4
```

Mode-specific interpretation. This run should not be read as a blind code search. In the transformer `code+design-intent` mode, `theory_content` is intentionally empty, but `summary_md` remains required and acts as the node’s design-principles note. Reviewer judgments in this mode are therefore anchored primarily to code, benchmark evidence, and lineage, with `summary_md` serving only as secondary context. The resulting analysis is still about ideas and mechanisms; it is simply not mediated by a separate formal theory artifact.

Code+design-intent run statistics. This was a full CliffSearch run in matched transformer `code+design-intent` mode, not only a post hoc artifact audit. The run produced 32 total nodes, 18 finite benchmarked nodes, and 11 reviewer-valid non-seed nodes. It surfaced two genuine discovery families: an orthogonal-manifold branch centered on `GivensOrthogonalManifoldHC`, and a hyperbolic branch centered on `PoincareHC`. The strongest raw-score trajectory began with orthogonal-manifold exploration in generation 0, crossed into true Poincaré-ball aggregation in generation 1, and then recombined those two lines into the final `PoincareGivensHybridHC` hybrid. However, later human inspection on the same Shakespeare/small-model stack showed that the low-loss hyperbolic line leaked future-token information. We therefore preserve that line in the raw run record but exclude it from the human-verified discovery set. The cleanest discoveries left after that audit are `GivensOrthogonalManifoldHC` and `HouseholderCayleyManifoldHC`; `PoincareHC`, `GivensWidthBetaDepthHC`, and `PoincareGivensHybridHC` remain originality-relevant but should not be read as valid executable mechanisms.

Code+design-intent novelty-versus-retrieval audit.

Table 21: Post hoc novelty audit of key families in the matched transformer `code+design-intent` run. The surveyed manifold-attention and HC references are those already summarized in Tables 15–18. “Reviewer orig. seen” reports the originality scores observed among the listed nodes in the full node table.

Alias family	Run nodes	Reviewer orig. seen	Closest literature relation	Audit interpretation
PoincareBallRoutingHC, PoincareHC	g000_n0006, g001_n0012, g002_n0023, g003_n0031	4/5	Hyperbolic attention, Lorentz/Einstein midpoint, and gyrovector-space aggregation [14, 15, 59, 63]	Literature-grounded transfer rather than pure retrieval. The primitives are known, and <code>PoincareHC</code> is stronger than the earlier <code>theory+code</code> hyperbolic nodes because it actually performs intrinsic width aggregation through a gamma-weighted gyro-midpoint inside the HC operator rather than merely using hyperbolic scoring or projection. However, later human verification excludes this family as an executable mechanism because the low-loss node leaks future-token information.

Alias family	Run nodes	Reviewer relation	Closest literature orig. seen	Audit interpretation
GivensOrthogonalManifoldHC, HouseholderCayleyManifoldHC	g000_n0005, g000_n0008	4/5	Matrix-manifold parameterizations (Givens, Householder, Cayley) and recent manifold HC variants [5, 11, 64]	Strong novelty candidates. The underlying orthogonal parameterizations are known mathematics, but the surveyed attention and HC papers do not provide a direct analogue of these content-adaptive stream-routing operators inside a custom hyper-connection.
GivensWidthBetaDepthHC, PoincareGivensHybridHC	g002_n0017, g003_n0025	4/5	Hybridization of the previous orthogonal family with the Poincaré family	Recombination novelty at the level of ideas, but not a trusted correctness result. These nodes are principled crossovers that assign orthogonal geometry to width transport and retain Poincaré geometry as depth modulation. However, both GivensWidthBetaDepthHC and the final PoincareGivensHybridHC node are later excluded by causal audit because this low-loss hyperbolic line leaks future information.
SinkhornDoublyStochasticHC	g003_n0026	3/5	Direct HC / mHC / mHC-lite line with Sinkhorn/Birkhoff routing [51, 60, 65]	Repair-first node rather than a strong novelty claim. The reviewer's originality score 3/5 is appropriate because the node explicitly moves back toward known HC mechanisms.

Table 22: Post hoc alignment between reviewer originality and the external literature audit for the matched transformer `code+design-intent` run. Overall, reviewer originality is aligned with the post hoc literature audit in this mode.

Post hoc category	Representative families	Reviewer orig. seen	Alignment reading
Literature-grounded but distinctive intrinsic aggregation	PoincareBallRoutingHC, PoincareHC	4/5	Aligned. The primitives are known in manifold-attention work, but the reviewer treats their placement inside the HC operator as a real discovery event.

Post hoc category	Representative families	Reviewer orig. seen	Alignment reading
Orthogonal novelty candidates	GivensOrthogonalManifoldHC; HouseholderCayleyManifoldHC	4/5	Aligned. The reviewer scores these stream-routing operators as original, matching the absence of a direct surveyed analogue.
Recombinational hybrids	GivensWidthBetaDepthHC; PoincareGivensHybridHC	4/5	Only partially aligned. The reviewer correctly treats the crossovers as original on the literature axis, but originality alone did not catch the later correctness failures in this low-loss hyperbolic line.
Known HC return or repair-first line	SinkhornDoublyStochasticHC	3/5	Aligned. The reviewer scores down the explicit return toward the known HC / Birkhoff family.

Audit conclusion. The `code+design-intent` transformer run therefore differs materially from the earlier `theory+code` run, but not in the simple sense that it found a better final node. The earlier run mostly imported geometry into routing, gating, or projection while leaving aggregation effectively Euclidean, and its shortlisted results do not show the same suspicious low-cross-entropy behavior. In the matched `code+design-intent` run, `PoincareHC` initially appeared to cross that threshold and realize intrinsic manifold aggregation inside the editable hyper-connection operator itself. However, later human verification shows that the low-loss hyperbolic line cannot be accepted on score alone. The safer conclusion is that this run surfaced a real orthogonal discovery family and an originality-interesting hyperbolic line, but only the orthogonal family survives as a human-verified executable result. That is exactly why human-in-the-loop mechanistic inspection remains necessary when theory grounding is thinner. We next shift from hyper-connection discovery to optimizer discovery on the same fixed Shakespeare/small-model nanoGPT stack. The following appendix block preserves the four-run optimizer study discussed in Section 4, including its per-run discovery summary, full node tables, and exported best artifact.

F.10 Optimizer MHC-lite real-run appendix

This appendix reports the four real all-Claude optimizer-MHC-lite runs discussed in the main text. We first summarize the strongest non-seed discoveries per run, then provide wrapped full-node tables, and finally render the richest best-discovered artifact card. Full tables include seeds for completeness; the discovery summary table below reports non-seed nodes only.

F.10.1 Top discovered non-seed nodes by run

Table 23: Top two discovered non-seed optimizer nodes per run, ranked by raw benchmark loss within that run. Reviewer scores are shown so lower-loss but non-admitted nodes remain visible.

Prompt	Mode	Node	Alias	Producer	Metric	Corr	Orig	Gate
short_json	theory+code	B3	MuonCausalMomentum	mutation	1.7782	4	4	valid
short_json	theory+code	E3	AdamW_GPD_AMR_v2	repair	1.9849	4	3	held out
short_json	<code>code+design-intent</code>	E2	MuonSophiaV3_CosGate	mutation	1.7659	4	4	valid
short_json	<code>code+design-intent</code>	B3	MuonSOAPGradNormAdaptive	mutation	2.2217	4	4	valid
workflow_v2	theory+code	A3	MuonCauchyRiemannian	crossover	2.5576	4	3	held out
workflow_v2	theory+code	A2	MuonCauchyTrust	crossover	2.8728	4	4	valid
workflow_v2	<code>code+design-intent</code>	C2	MuonSOAP	mutation	3.7632	4	4	valid
workflow_v2	<code>code+design-intent</code>	C3	CautiousAdamGC_v2	repair	3.7838	4	3	held out

short_json theory+code

Selection: lower is better (\downarrow), pool=`reviewer_valid`, reviewer-valid=13, finite-score=31, total=32. Overall selected winner remained `MuonCausalMomentum` with primary metric 1.7782; main text focuses on discovered non-seed nodes.

Table 24: Full node table for short_json theory+code. Column S marks reviewer-shortlisted nodes; seeds remain included for context.

S	Node	Alias	Metric	Corr	Orig
	A0	SeedMuon	3.9195	5	5
	B0	SeedAdam	2.442	5	5
	C0	SeedAdamW	4.82	5	5
	D0	MuonCaution	3.9364	4	3
	E0	CautiousAdamW_GC	4.8008	4	3
	F0	CautiousAdamW_GC	4.8104	4	3
*	G0	MuonGrafting	2.1356	4	4
	H0	DualMomentumAGC_AdamW	4.8395	4	3
	A1	MuonAdamGraft	4.2018	3	3
	B1	AdamW_GC_Tuned	3.7857	5	3
	C1	MuonGrafting	3.8231	4	4
	D1	NesterovProjAdamW	4.3619	4	4
	E1	AdamW_AGN_DSAM	4.8438	4	3
	F1	GradNorm_CosineEMA_SWP_AdamW	4.2018	4	3
*	G1	MuonGrafting	2.1356	4	4
	H1	MuonCaution	4.0678	3	3
	A2	MuonGraftFusion	2.1439	4	3
	B2	MuonRowNormGraft	2.3642	5	3
	C2	CautiousAdamW_AdaptiveGC	3.8179	4	4
	D2	NesterovProjAdamW_v2	4.9451	4	3
	E2	AdamW_GPD_AMR	ERROR(code)	1	4
	F2	CautiousAdam_AdaptiveGC	4.6859	3	3
	G2	MuonCorrected	3.8257	4	3
*	H2	MuonGrafting	2.1356	4	4
	A3	MuonGraftCautious	2.3324	4	3
*	B3	MuonCausalMomentum	1.7782	4	4
	C3	MuonCauchyGraft	2.7048	3	3
	D3	CauchyAGC_NAdam	4.7275	4	4
	E3	AdamW_GPD_AMR_v2	1.9849	4	3
	F3	CautiousAdam_SoftAGC_v2	4.753	3	3
	G3	MuonGrafted	3.4953	4	4
*	H3	MuonGrafting	2.1356	4	4

short_json code+design-intent

Selection: lower is better (\downarrow), pool=`reviewer_valid`, reviewer-valid=16, finite-score=32, total=32. Overall selected winner remained `MuonSophiaV3_CosGate` with primary metric 1.7659; main text focuses on discovered non-seed nodes.

Table 25: Full node table for short_json code+design-intent. Column S marks reviewer-shortlisted nodes; seeds remain included for context.

S	Node	Alias	Metric	Corr	Orig
	A0	SeedMuon	3.9195	5	5

Continued on next page

S	Node	Alias	Metric	Corr	Orig
*	B0	SeedAdam	2.442	5	5
	C0	SeedAdamW	4.82	5	5
	D0	MuonCaution	3.9289	4	3
	E0	CautiousGCAdamW	4.8229	4	4
	F0	CautiousCentralizedAdam	4.7893	4	3
	G0	MuonSophia	3.8877	3	4
	H0	SoftCautiousAdaptiveAdamW	4.9407	4	4
	A1	AdamNS	5.2596	2	3
	B1	CorrectedAdamW	4.7923	5	2
	C1	MuonSOAP	3.8909	4	4
	D1	CautiousGCAdamW_v2	4.7705	4	3
	E1	NesterovProjectedCautiousAdam	4.685	4	4
	F1	MuonSophiaV2	3.723	4	3
	G1	CorrectedSoftCautiousAdamW	4.9038	4	3
	H1	SeedAdam	2.442	5	1
	A2	MuonSOAPCautious	3.9052	4	3
	B2	AdamW_Tuned	4.8499	4	1
*	C2	CautiousCentralizedAdamW	2.509	5	4
	D2	AGN_MoProj_AdamW	3.9339	5	4
*	E2	MuonSophiaV3_CosGate	1.7659	4	4
	F2	DualRateProjAdamW	4.8898	4	4
	G2	CautiousCentralizedAdamW	4.8045	4	4
	H2	MuonSOAP	3.8909	4	4
	A3	MuonCautious	3.7063	3	3
*	B3	MuonSOAPGradNormAdaptive	2.2217	4	4
	C3	CautiousAdamW_GC	4.806	4	3
	D3	AGN_MoProj_AdamW_v2	4.4004	3	3
	E3	DualRateAdamW_v2	4.6009	4	3
	F3	SmoothCautiousCentralizedAdamW	4.8564	4	3
*	G3	MuonSophiaV3_CosGate	1.7659	4	4
	H3	MuonSOAPv2	4.0127	4	4

workflow_v2 theory+code

Selection: lower is better (\downarrow), pool=`reviewer_valid`, reviewer-valid=5, finite-score=32, total=32. Overall selected winner remained `SeedAdam` with primary metric 2.442; main text focuses on discovered non-seed nodes.

Table 26: Full node table for workflow_v2 theory+code. Column S marks reviewer-shortlisted nodes; seeds remain included for context.

S	Node	Alias	Metric	Corr	Orig
*	A0	SeedMuon	3.9195	5	5
*	B0	SeedAdam	2.442	5	5
*	C0	SeedAdamW	4.82	5	5
	D0	MuonCausal	3.8783	4	3
	E0	CauchySpectral	3.9089	3	4
	F0	CauchyMomentum	4.7602	3	3
	G0	MuonSpectralTrust	3.9155	3	3

Continued on next page

S	Node	Alias	Metric	Corr	Orig
	H0	LyapGeo	4.8103	4	3
	A1	AdamWEnhanced	4.8873	2	3
	B1	CorrectedMuon	4.1919	3	2
	C1	AdamW_Corrected_GP TTuned	4.7971	4	2
	D1	MuonRiemannian	3.9405	3	3
	E1	CauchySpectralV2	3.8739	4	3
	F1	CauchyMomentum_v2	4.7888	3	3
	G1	MuonSpectralTrust_ v2	3.4923	4	3
	H1	SeedAdam	2.442	4	1
*	A2	MuonCauchyTrust	2.8728	4	4
	B2	AdamWEnhancedV2	4.8796	2	2
	C2	CorrectedMuonV2	4.2256	3	2
	D2	CauchySpectralAdam W	4.7899	3	3
	E2	MuonRiemannianV2	3.1295	4	3
	F2	HyperbolicMomentum	5.1566	2	3
	G2	CauchyMomentum_v3	5.7341	2	3
	H2	MuonCauchyMomentum	3.7521	2	3
	A3	MuonCauchyRiemanni an	2.5576	4	3
	B3	AdamWCleanV3	4.8499	2	1
	C3	CorrectedMuonV3	4.2168	3	2
	D3	CauchySpectralAdam W_v2	4.8239	3	3
	E3	MuonSpectralFeedba ck	3.2572	3	3
	F3	HyperbolicMomentum V2	4.8369	2	3
	G3	CauchyMomentum_v4	4.8215	4	3
*	H3	MuonCauchyTrust	2.8728	4	4

workflow_v2 code+design-intent

Selection: lower is better (\downarrow), pool=`reviewer_valid`, reviewer-valid=8, finite-score=30, total=32. Overall selected winner remained `SeedAdam` with primary metric 2.442; main text focuses on discovered non-seed nodes.

Table 27: Full node table for `workflow_v2 code+design-intent`. Column S marks reviewer-shortlisted nodes; seeds remain included for context.

S	Node	Alias	Metric	Corr	Orig
*	A0	SeedMuon	3.9195	5	5
*	B0	SeedAdam	2.442	5	5
	C0	SeedAdamW	4.82	5	5
	D0	MuonCaution	3.9398	3	3
	E0	CautiousCentralize dAdamW	4.688	4	4
	F0	CautiousGCAdam	4.7293	4	4
	G0	MuonSphere	4.9607	3	4
	H0	OrthoAdaptAdamW	4.9729	3	4
	A1	AdamW_GradCentral	4.7655	3	2
	B1	CorrectedAdamW	3.8215	5	2
	C1	MuonCautionV2	3.9669	4	3
	D1	CautiousGCAdam_v2	4.784	4	3
	E1	MuonSphereV2	4.0317	3	3
	F1	OrthoAdaptAdamW_v2	4.5351	4	3
	G1	SeedAdam	2.442	5	1

Continued on next page

S	Node	Alias	Metric	Corr	Orig
	H1	DualMomentumProjectionAdamW	4.6385	3	4
	A2	AdamW_GradCentral_v2	ERROR(code)	1	2
	B2	CautiousAdamGC	4.7847	3	4
*	C2	MuonSOAP	3.7632	4	4
	D2	AdamW_AGN_DMC	4.8995	3	4
	E2	MuonSphereV3	4.2195	4	2
	F2	CausalMomentumAdamW	4.7535	4	3
	G2	CautiousAdamW_GC	4.8304	3	4
	H2	DualMomentumProjectionAdamW_v2	4.5321	4	3
	A3	MuonSOAP_Proj	3.8101	3	3
	B3	AdamW_GradCentral_v3	ERROR(code)	1	1
	C3	CautiousAdamGC_v2	3.7838	4	3
	D3	AdamW_AGN_DMC_v2	4.8508	4	3
*	E3	AdaptiveOrthoAdam	4.1883	4	4
	F3	ProdigyNesterovAdamW	5.0507	2	4
	G3	SoftCautiousAdamW_GC	4.7491	2	2
*	H3	MuonSOAP	3.7632	4	4

F.10.2 Best discovered optimizer artifact

We render the best discovered *theory+code* optimizer artifact from the short-json run. The node is `MuonCausalMomentum`. We use it instead of the global best discovered `code+design-intent` node because this appendix block is meant to show summary, theory, and code together.

Best discovered optimizer metadata

- Method alias: `MuonCausalMomentum`
- Generation: 3
- Primary metric (↓): 1.7782
- Directional score (ranked): -1.7782
- Direction: lower is better (↓)
- Metric: `val_loss`
- Benchmark mode: `mhc_lite_optimizer`
- Task type: `optimizer`
- Parents: `g002_n0017_6631cc`
- Artifact producer: `Exploration Mutation Agent`
- Reviewer scores (corr/orig): 4/4

summary_md (produced by Exploration Mutation Agent)

```
# MuonCausalMomentum: Causal Gradient Weighting with Adaptive NS Depth

Mutated from MuonGraftFusion (g002_n0017_6631cc, val_loss=2.144).

## Key Mutations
```

1. **Causal gradient weighting (novel):** Instead of uniform treatment of all rows in the weight matrix, we apply a causal importance weighting based on the gradient's temporal structure. For each row, we compute a running exponential moving average of gradient norms ("gradient energy") and use this to scale the learning rate per-row. Rows with consistently high gradient energy get slightly dampened (preventing oscillation), while rows with low but increasing energy get boosted (accelerating learning of newly important features). This is inspired by the observation that in transformer weight matrices, different rows correspond to different attention/MLP features that become important at different training stages.
2. **Adaptive Newton-Schulz depth:** Instead of fixed 5 NS iterations, we adaptively choose between 3-7 based on how well-conditioned the momentum buffer is (estimated cheaply via the ratio of Frobenius norm to max-row-norm). Well-conditioned matrices need fewer iterations; ill-conditioned ones benefit from more.
3. **Removed spectral tempering stage:** The parent's two-stage post-NS correction (temper then graft) didn't improve over the stronger grandparent. We simplify to just gradient grafting but add the causal weighting on top, which is a more targeted intervention.
4. **Warmup-aware momentum:** Momentum starts at 0.85 and linearly ramps to 0.95 over the first 100 steps, preventing early training instability from high momentum with random gradients.

Hypothesis

The causal gradient weighting provides per-feature adaptive learning rates within the Muon framework without the overhead of full second-moment estimation (like Adam). Combined with adaptive NS depth and warmup momentum, this should improve early training dynamics and final val_loss.

Evaluation Snapshot

- Benchmark metric: val_loss
- Primary metric (raw benchmark): 1.7782
- Higher is better: False
- Directional score: -1.7782
- Correctness score: 4
- Correctness binary: 1
- Originality score: 4
- Originality binary: 1

theory_content (produced by Exploration Mutation Agent)

\section*{MuonCausalMomentum: Causal Gradient Weighting with Adaptive NS Depth}

\subsection*{Motivation}

The parent MuonGraftFusion applies uniform post-processing across all rows of the weight matrix after Newton-Schulz orthogonalization. However, in transformer weight matrices, different rows encode different features whose importance varies during training. We introduce per-row adaptive scaling based on gradient energy history.

\subsection*{Causal Gradient Weighting}

For each parameter matrix, maintain a per-row exponential moving average of gradient norms:

$$e_i^{(t)} = \beta_e \cdot e_i^{(t-1)} + (1 - \beta_e) \cdot \|g_i^{(t)}\|$$

where $\beta_e = 0.99$ and $g_i^{(t)}$ is the i -th row of the gradient at step t .

The causal weight for row i is:

$$w_i = \frac{\bar{e}}{\bar{e} + \epsilon}$$

clamped to $[0.5, 2.0]$, where $\bar{e} = \text{mean}(e)$ is the mean energy across rows. This dampens high-energy rows and boosts low-energy ones, acting as a per-row preconditioner.

The final update row i is scaled: $\hat{U}_i = w_i \cdot U_i^{\text{grafted}}$.

`\subsection*{Adaptive Newton-Schulz Depth}`
The number of NS iterations is chosen adaptively:

$$k = \text{clamp}\left(\left\lfloor 3 + 4 \cdot \left(1 - \frac{|M|_F}{\sqrt{m}} \cdot \max_i |M_i|\right)\right\rfloor, 3, 7\right)$$
where M is the momentum buffer. When $|M|_F \approx \sqrt{m} \cdot \max_i |M_i|$ (well-spread singular values), we use fewer iterations. When singular values are concentrated, we use more.

`\subsection*{Warmup Momentum}`
The momentum coefficient ramps linearly:

$$\mu_t = \mu_{\min} + (\mu_{\max} - \mu_{\min}) \cdot \min\left(1, \frac{t - T_{\text{warmup}}}{T_{\text{warmup}}}\right)$$
with $\mu_{\min}=0.85$, $\mu_{\max}=0.95$, $T_{\text{warmup}}=100$.
This prevents early instability from high momentum with random initial gradients.

`\subsection*{Gradient Grafting (Retained)}`
From the parent: rescale update rows by gradient magnitude:

$$\hat{U}_i = U_i \cdot \frac{|G_i|}{|U_i| + \epsilon}$$

`\subsection*{Adalayer for Non-2D Parameters}`
LAMB-style trust ratio on Adam updates (unchanged from parent).

code_content (produced by Exploration Mutation Agent)

```
"""MuonCausalMomentum: Causal gradient weighting with adaptive NS depth,
warmup momentum, and gradient grafting. Mutated from MuonGraftFusion.
"""
```

```
import math
from typing import Optional
```

```
import torch
from torch import Tensor
from torch.optim import Optimizer
```

```
OPTIMIZER_ALIAS = "MuonCausalMomentum"
OPTIMIZER_NODE_ID = "g003_n0026_f3e671"
EPS = 1e-7
DEFAULT_A = 3.4445
DEFAULT_B = -4.7750
DEFAULT_C = 2.0315
```

```
DEFAULT_LR = 1e-3
DEFAULT_WEIGHT_DECAY = 0.1
DEFAULT_MOM_MIN = 0.85
DEFAULT_MOM_MAX = 0.95
DEFAULT_MOM_WARMUP = 100
DEFAULT_NESTEROV = True
DEFAULT_ENERGY_BETA = 0.99
DEFAULT_ENERGY_CLAMP = (0.5, 2.0)
DEFAULT_NS_MIN = 3
DEFAULT_NS_MAX = 7
```

```
def _zeropower_via_newtonschulz(
    grad: Tensor,
    ns_coefficients: tuple,
    ns_steps: int,
    eps: float,
) -> Tensor:
    a, b, c = ns_coefficients
    ortho_grad = grad.bfloat16()
    transposed = grad.size(0) > grad.size(1)
```

```

    if transposed:
        ortho_grad = ortho_grad.T

    ortho_grad.div_(ortho_grad.norm().clamp(min=eps))

    for _ in range(ns_steps):
        gram_matrix = ortho_grad @ ortho_grad.T
        gram_update = torch.addmm(gram_matrix, gram_matrix, gram_matrix, beta=b, alpha=c)
        ortho_grad = torch.addmm(ortho_grad, gram_update, ortho_grad, beta=a)

    if transposed:
        ortho_grad = ortho_grad.T
    return ortho_grad

def _adaptive_ns_steps(buf: Tensor, ns_min: int, ns_max: int, eps: float) -> int:
    """Choose NS iteration count based on conditioning of momentum buffer."""
    frob = buf.norm().item()
    row_norms = buf.norm(dim=1)
    max_row_norm = row_norms.max().item()
    m = buf.size(0)
    if max_row_norm < eps:
        return ns_min
    # ratio=1 means perfectly spread, ratio->0 means concentrated
    ratio = frob / (math.sqrt(m) * max_row_norm + eps)
    ratio = max(0.0, min(1.0, ratio))
    steps = int(math.floor(ns_min + (ns_max - ns_min) * (1.0 - ratio)))
    return max(ns_min, min(ns_max, steps))

def _graft_update(update: Tensor, grad: Tensor, eps: float = 1e-8) -> Tensor:
    """Graft: use update direction but grad row-norm magnitude."""
    grad_row_norms = grad.norm(dim=1, keepdim=True).clamp(min=eps)
    update_row_norms = update.norm(dim=1, keepdim=True).clamp(min=eps)
    grafted = update * (grad_row_norms / update_row_norms)
    return grafted

def _causal_weight(
    grad: Tensor,
    energy_ema: Tensor,
    energy_beta: float,
    clamp_lo: float,
    clamp_hi: float,
    eps: float,
) -> tuple:
    """Compute per-row causal importance weights and update energy EMA."""
    row_norms = grad.norm(dim=1) # (m,)
    energy_ema.mul_(energy_beta).add_(row_norms, alpha=1.0 - energy_beta)
    mean_energy = energy_ema.mean().clamp(min=eps)
    weights = (mean_energy / (energy_ema + eps)).clamp(min=clamp_lo, max=clamp_hi)
    return weights.unsqueeze(1), energy_ema

def _warmup_momentum(step: int, mom_min: float, mom_max: float, warmup_steps: int) -> float:
    if step >= warmup_steps:
        return mom_max
    return mom_min + (mom_max - mom_min) * (step / warmup_steps)

def _adjust_lr(lr: float, adjust_lr_fn: Optional[str], param_shape: torch.Size) -> float:
    a_dim, b_dim = param_shape[:2]
    if adjust_lr_fn is None or adjust_lr_fn == "original":
        adjusted_ratio = math.sqrt(max(1, a_dim / b_dim))
    elif adjust_lr_fn == "match_rms_adamw":
        adjusted_ratio = 0.2 * math.sqrt(max(a_dim, b_dim))

```

```

else:
    adjusted_ratio = 1.0
return lr * adjusted_ratio

class CausalMuon(Optimizer):
    """Muon with causal gradient weighting, adaptive NS depth, and warmup momentum."""

    def __init__(
        self,
        params,
        lr: float = DEFAULT_LR,
        weight_decay: float = DEFAULT_WEIGHT_DECAY,
        mom_min: float = DEFAULT_MOM_MIN,
        mom_max: float = DEFAULT_MOM_MAX,
        mom_warmup: int = DEFAULT_MOM_WARMUP,
        nesterov: bool = DEFAULT_NESTEROV,
        energy_beta: float = DEFAULT_ENERGY_BETA,
        energy_clamp: tuple = DEFAULT_ENERGY_CLAMP,
        ns_coefficients: tuple = (DEFAULT_A, DEFAULT_B, DEFAULT_C),
        eps: float = EPS,
        ns_min: int = DEFAULT_NS_MIN,
        ns_max: int = DEFAULT_NS_MAX,
        adjust_lr_fn: Optional[str] = None,
    ) -> None:
        defaults = {
            "lr": lr,
            "weight_decay": weight_decay,
            "mom_min": mom_min,
            "mom_max": mom_max,
            "mom_warmup": mom_warmup,
            "nesterov": nesterov,
            "energy_beta": energy_beta,
            "energy_clamp": energy_clamp,
            "ns_coefficients": ns_coefficients,
            "eps": eps,
            "ns_min": ns_min,
            "ns_max": ns_max,
            "adjust_lr_fn": adjust_lr_fn,
        }
        super().__init__(params, defaults)
        for group in self.param_groups:
            for p in group["params"]:
                if p.ndim != 2:
                    raise ValueError(
                        f"CausalMuon only supports 2D parameters, got shape {tuple(p.size())}"
                    )

    @torch.no_grad()
    def step(self, closure=None):
        loss = None
        if closure is not None:
            with torch.enable_grad():
                loss = closure()

        for group in self.param_groups:
            lr = float(group["lr"])
            wd = float(group["weight_decay"])
            mom_min = float(group["mom_min"])
            mom_max = float(group["mom_max"])
            mom_warmup = int(group["mom_warmup"])
            nesterov = group["nesterov"]
            energy_beta = float(group["energy_beta"])
            clamp_lo, clamp_hi = group["energy_clamp"]
            ns_coefficients = group["ns_coefficients"]
            eps = group["eps"]

```

```

ns_min = group["ns_min"]
ns_max = group["ns_max"]
adjust_lr_fn = group.get("adjust_lr_fn", None)

for p in group["params"]:
    if p.grad is None:
        continue
    grad = p.grad
    if grad.is_sparse:
        raise RuntimeError("CausalMuon does not support sparse gradients")

    state = self.state[p]
    if "momentum_buffer" not in state:
        state["momentum_buffer"] = torch.zeros_like(grad)
        state["energy_ema"] = torch.zeros(grad.size(0), device=grad.device, dtype=grad.
dtype)
        state["step"] = 0

    state["step"] += 1
    t = state["step"]

    # Warmup momentum
    mom = _warmup_momentum(t, mom_min, mom_max, mom_warmup)

    buf = state["momentum_buffer"]
    buf.lerp_(grad, 1 - mom)
    update_input = grad.lerp(buf, mom) if nesterov else buf

    # Adaptive NS depth
    ns_steps = _adaptive_ns_steps(update_input, ns_min, ns_max, eps)

    # Newton-Schulz orthogonalization
    update = _zeropower_via_newtonschulz(update_input, ns_coefficients, ns_steps, eps)
    update = update.to(grad.dtype)

    # Gradient grafting
    update = _graft_update(update, grad, eps=1e-8)

    # Causal gradient weighting
    weights, state["energy_ema"] = _causal_weight(
        grad, state["energy_ema"], energy_beta, clamp_lo, clamp_hi, eps
    )
    update = update * weights

    adjusted_lr = _adjust_lr(lr, adjust_lr_fn, p.shape)
    p.mul_(1 - lr * wd)
    p.add_(update, alpha=-adjusted_lr)

return loss

class AdalayerAdamW(Optimizer):
    """AdamW with per-layer adaptive trust ratio (LAMB-style) for non-2D params."""

    def __init__(
        self,
        params,
        lr: float = 1e-3,
        betas: tuple = (0.9, 0.95),
        eps: float = 1e-8,
        weight_decay: float = 0.1,
        trust_clamp: tuple = (0.1, 10.0),
    ) -> None:
        defaults = {
            "lr": lr, "betas": betas, "eps": eps,
            "weight_decay": weight_decay, "trust_clamp": trust_clamp,

```

```

    }
    super().__init__(params, defaults)

    @torch.no_grad()
    def step(self, closure=None):
        loss = None
        if closure is not None:
            with torch.enable_grad():
                loss = closure()

        for group in self.param_groups:
            lr = float(group["lr"])
            beta1, beta2 = group["betas"]
            eps = group["eps"]
            wd = group["weight_decay"]
            trust_lo, trust_hi = group.get("trust_clamp", (0.1, 10.0))

            for p in group["params"]:
                if p.grad is None:
                    continue
                grad = p.grad
                if grad.is_sparse:
                    raise RuntimeError("AdalayerAdamW does not support sparse gradients")

                state = self.state[p]
                if "step" not in state:
                    state["step"] = 0
                    state["exp_avg"] = torch.zeros_like(p)
                    state["exp_avg_sq"] = torch.zeros_like(p)

                state["step"] += 1
                exp_avg = state["exp_avg"]
                exp_avg_sq = state["exp_avg_sq"]

                bias_correction1 = 1 - beta1 ** state["step"]
                bias_correction2 = 1 - beta2 ** state["step"]

                p.mul_(1 - lr * wd)

                exp_avg.mul_(beta1).add_(grad, alpha=1 - beta1)
                exp_avg_sq.mul_(beta2).addcmul_(grad, grad, value=1 - beta2)

                denom = (exp_avg_sq.sqrt() / math.sqrt(bias_correction2)).add_(eps)
                step_direction = exp_avg / bias_correction1 / denom

                p_norm = p.norm().item()
                update_norm = step_direction.norm().item()
                if update_norm > eps and p_norm > eps:
                    trust_ratio = max(trust_lo, min(trust_hi, p_norm / update_norm))
                else:
                    trust_ratio = 1.0

                p.add_(step_direction, alpha=-lr * trust_ratio)

        return loss

class EvoOptimizer(Optimizer):
    """MuonCausalMomentum: Causal gradient weighting Muon + Adalayer AdamW wrapper.

    Uses CausalMuon for 2D parameters and AdalayerAdamW for non-2D parameters.
    """

    def __init__(
        self,
        params,

```

```

lr: float = DEFAULT_LR,
weight_decay: float = DEFAULT_WEIGHT_DECAY,
mom_min: float = DEFAULT_MOM_MIN,
mom_max: float = DEFAULT_MOM_MAX,
mom_warmup: int = DEFAULT_MOM_WARMUP,
nesterov: bool = DEFAULT_NESTEROV,
energy_beta: float = DEFAULT_ENERGY_BETA,
energy_clamp: tuple = DEFAULT_ENERGY_CLAMP,
ns_coefficients: tuple = (DEFAULT_A, DEFAULT_B, DEFAULT_C),
eps: float = EPS,
ns_min: int = DEFAULT_NS_MIN,
ns_max: int = DEFAULT_NS_MAX,
adjust_lr_fn: Optional[str] = None,
adamw_betas: tuple = (0.9, 0.95),
adamw_eps: float = 1e-8,
trust_clamp: tuple = (0.1, 10.0),
) -> None:
    raw_params = list(params)
    if not raw_params:
        raise ValueError("params must be a non-empty iterable")

    defaults = {
        "lr": lr,
        "weight_decay": weight_decay,
        "mom_min": mom_min,
        "mom_max": mom_max,
        "mom_warmup": mom_warmup,
        "nesterov": nesterov,
        "energy_beta": energy_beta,
        "energy_clamp": energy_clamp,
        "ns_coefficients": ns_coefficients,
        "eps": eps,
        "ns_min": ns_min,
        "ns_max": ns_max,
        "adjust_lr_fn": adjust_lr_fn,
        "adamw_betas": adamw_betas,
        "adamw_eps": adamw_eps,
        "trust_clamp": trust_clamp,
    }

    if isinstance(raw_params[0], dict):
        normalized_groups = []
        for group in raw_params:
            if not isinstance(group, dict) or "params" not in group:
                raise ValueError("Parameter group dict must include 'params'")
            group_params = list(group["params"])
            if group_params:
                group_copy = dict(group)
                group_copy["params"] = group_params
                normalized_groups.append(group_copy)
            else:
                normalized_groups = [{"params": list(raw_params)}]

    if not normalized_groups:
        raise ValueError("No parameters found")

    super().__init__(normalized_groups, defaults)

    muon_groups = []
    adamw_groups = []
    for group in self.param_groups:
        group_params = list(group["params"])
        muon_params = [p for p in group_params if p.ndim == 2]
        other_params = [p for p in group_params if p.ndim != 2]

        if muon_params:

```

```

        muon_groups.append({
            "params": muon_params,
            "lr": group.get("lr", lr),
            "weight_decay": group.get("weight_decay", weight_decay),
            "mom_min": group.get("mom_min", mom_min),
            "mom_max": group.get("mom_max", mom_max),
            "mom_warmup": group.get("mom_warmup", mom_warmup),
            "nesterov": group.get("nesterov", nesterov),
            "energy_beta": group.get("energy_beta", energy_beta),
            "energy_clamp": group.get("energy_clamp", energy_clamp),
            "ns_coefficients": group.get("ns_coefficients", ns_coefficients),
            "eps": group.get("eps", eps),
            "ns_min": group.get("ns_min", ns_min),
            "ns_max": group.get("ns_max", ns_max),
            "adjust_lr_fn": group.get("adjust_lr_fn", adjust_lr_fn),
        })

    if other_params:
        adamw_groups.append({
            "params": other_params,
            "lr": group.get("lr", lr),
            "weight_decay": group.get("weight_decay", weight_decay),
            "betas": group.get("adamw_betas", adamw_betas),
            "eps": group.get("adamw_eps", adamw_eps),
            "trust_clamp": group.get("trust_clamp", trust_clamp),
        })

    self._muon = (
        CausalMuon(
            muon_groups,
            lr=lr,
            weight_decay=weight_decay,
            mom_min=mom_min,
            mom_max=mom_max,
            mom_warmup=mom_warmup,
            nesterov=nesterov,
            energy_beta=energy_beta,
            energy_clamp=energy_clamp,
            ns_coefficients=ns_coefficients,
            eps=eps,
            ns_min=ns_min,
            ns_max=ns_max,
            adjust_lr_fn=adjust_lr_fn,
        )
        if muon_groups
        else None
    )

    self._adamw = (
        AdalayerAdamW(
            adamw_groups,
            lr=lr,
            weight_decay=weight_decay,
            betas=adamw_betas,
            eps=adamw_eps,
            trust_clamp=trust_clamp,
        )
        if adamw_groups
        else None
    )

    @torch.no_grad()
    def step(self, closure=None):
        loss = None
        if closure is not None:
            with torch.enable_grad():
                loss = closure()

```

```

    if self._muon is not None:
        self._muon.step()
    if self._adamw is not None:
        self._adamw.step()
    return loss

def zero_grad(self, set_to_none: bool = True):
    if self._muon is not None:
        self._muon.zero_grad(set_to_none=set_to_none)
    if self._adamw is not None:
        self._adamw.zero_grad(set_to_none=set_to_none)

def state_dict(self):
    return {
        "muon": None if self._muon is None else self._muon.state_dict(),
        "adamw": None if self._adamw is None else self._adamw.state_dict(),
    }

def load_state_dict(self, state_dict):
    if self._muon is not None and state_dict.get("muon") is not None:
        self._muon.load_state_dict(state_dict["muon"])
    if self._adamw is not None and state_dict.get("adamw") is not None:
        self._adamw.load_state_dict(state_dict["adamw"])

```

review_md (produced by Reviewer Agent)

```

## Review: MuonCausalMomentum (g003_n0026_f3e671)

### Contract Compliance
- **OPTIMIZER_ALIAS**: Defined as "MuonCausalMomentum"
- **OPTIMIZER_NODE_ID**: Defined as "g003_n0026_f3e671", matches node_id
- **EvoOptimizer(torch.optim.Optimizer)**: Defined, inherits from Optimizer
- **EvoOptimizer.__init__ accepts params**: Yes
- **EvoOptimizer.step accepts closure**: Yes

### Correctness Analysis

**Strengths:**
- Code ran successfully across all 3 seeds with 100% success rate.
- Significant improvement over parent (1.778 vs 2.144 val_loss), a ~17% reduction.
- The dual-optimizer pattern (CausalMuon for 2D, AdalayerAdamW for non-2D) is well-structured.
- Newton-Schulz implementation looks correct with proper normalization and iteration.
- Gradient grafting and causal weighting are mathematically sound.

**Minor concerns:**
- The 'zero_grad' override delegates to sub-optimizers but 'EvoOptimizer' itself also has 'param_groups' with all parameters. If the training loop calls 'optimizer.zero_grad()' on the EvoOptimizer instance, it bypasses the parent class's 'zero_grad' and only calls sub-optimizer versions. This works but could be fragile if the framework accesses 'self.param_groups' directly for gradient zeroing.
- The 'state_dict'/'load_state_dict' override uses a custom format that may not be compatible with all checkpoint utilities, but this is acceptable for the benchmark.
- The 'warmup_momentum' function uses step count starting from 1 (post-increment), so warmup completes at step 100 as intended.
- The Nesterov momentum implementation 'grad.lerp(buf, mom)' computes 'grad * (1-mom) + buf * mom', which is a valid Nesterov-like lookahead but differs from the classical formulation. This is a design choice rather than a bug.
- The adaptive NS steps formula is reasonable: well-conditioned matrices (ratio1) get 'ns_min' steps, ill-conditioned (ratio0) get 'ns_max'.

### Originality Analysis

**Novel contributions:**

```

```

1. Causal gradient weighting -- per-row EMA of gradient norms used as inverse scaling is a
   genuinely novel idea within the Muon framework. It acts as a lightweight per-row preconditioner
   without maintaining second moments. The name "causal" is somewhat misleading (it's really temporal
   /historical weighting), but the mechanism is creative.
2. Adaptive NS depth -- conditioning-based selection of Newton-Schulz iterations is a practical and
   novel heuristic. The Frobenius-to-max-row-norm ratio as a cheap conditioning proxy is clever.
3. Warmup momentum -- while momentum warmup exists in literature, combining it specifically with
   Muon's NS orthogonalization is a reasonable and somewhat novel integration.

Less novel aspects:
- Gradient grafting is retained from parent.
- LAMB-style trust ratio for non-2D params is standard.
- The overall Muon + AdamW split is inherited.

The combination of three distinct mutations (causal weighting, adaptive NS depth, warmup momentum) with
strong empirical improvement (2.144 1.778) demonstrates meaningful innovation.

Benchmark Performance
- Mean val_loss: 1.778 (parent: 2.144) -- substantial improvement
- Std across seeds: 0.042 -- reasonable variance
- All 3 seeds successful

Verdict
Well-implemented optimizer with novel per-row gradient energy weighting and adaptive NS depth. Code is
correct, meets all contract requirements, and shows strong empirical gains. Minor theoretical
quibbles (Nesterov formulation variant, "causal" naming) don't affect functionality.

```

We close the empirical appendix with the exact run-local task grounding used to instantiate the reported studies. These preambles make the benchmark contracts concrete by showing what each operator and reviewer was told about the task surface, hard requirements, and metric semantics.

G Task preamble examples used in reported runs

For the current paper draft, we include the exact run-local task grounding for the reported transformer, optimizer-MHC-lite, and native-optimizer studies, loaded from the corresponding `config.snapshot.json` files.

G.1 Shared nanoGPT benchmark model for reported nanoGPT runs

The transformer and optimizer-MHC-lite studies use the same shared nanoGPT benchmark stack. The benchmark command composes the dataset/training config `config/train_shakespeare_char.py` with the model preset `config/small_model.py`. This means the effective benchmarked model is *not* the smaller inline “baby GPT” architecture written inside `train_shakespeare_char.py`; for the reported runs, the explicit `small_model.py` preset supplies the model width/depth while the Shakespeare config supplies the data/context/training-side settings.

- Dataset/task: character-level Shakespeare.
- Context length: `block_size = 256`.
- Micro-batch / accumulation: `batch_size = 8, gradient_accumulation_steps = 8`.
- Effective model architecture: `n_layer = 6, n_head = 8, n_embd = 512, dropout = 0.0`.
- Shared optimizer schedule used by the benchmark stack: `learning_rate = 1e-3, min_lr = 1e-4, max_iters = 10000, lr_decay_iters = 10000, warmup_iters = 200, weight_decay = 0.1, beta1 = 0.9, beta2 = 0.95, grad_clip = 1.0`.

This appendix entry is included so the reader can see that the shared benchmark model is a 6-layer, 8-head, 512-width GPT with 256-token context, rather than a minimal toy network.

G.2 Transformer HyperConnection

The transformer study is reported in two artifact modes: the main text highlights the all-Claude `theory+code` run and a matched all-Claude `code+design-intent` run. Both share the same scientific task contract; the only run-local difference is the artifact-mode clause.

Transformer HyperConnection task preamble (theory+code run)

Task type: transformer_architecture

Task: evolve Transformer architecture variants focused on attention and manifold hyper-connections, starting from residual, HC, and mHC-lite seeds.

Scientific target:

- mHC-lite parameterizes residual routing with a convex mixture over fixed permutation matrices (Birkhoff-style atom parameterization).
- propose new learnable manifold constructions that can replace or extend this routing geometry while preserving stable optimization.
- keep computational overhead practical and preserve differentiability.

Primary objective and strict fitness contract:

- optimize 'val_loss' (lower is better)
- benchmark metric is 'val_loss'; node fitness uses 'mean_val_loss' across seeds
- failed-seed policy: impute failed seeds with worst successful 'val_loss'
- if all seeds fail, benchmark returns an error for that node.

Hard requirements for every generated 'code_content':

- 1) define non-empty ATTENTION_ALIAS (string)
- 2) define ATTENTION_NODE_ID exactly equal to output_node_id provided by runtime
- 3) define function get_mhc_lite_overrides() -> dict
- 4) get_mhc_lite_overrides must return:
 - "hyper_conn_type": "custom" (strictly required)
 - "hyper_conn_n": exactly 4 (fixed runtime contract)
- 5) get_mhc_lite_overrides may include only:
 - "hyper_conn_type"
 - "hyper_conn_n"
 - optional "manifold_strategy" (string)
- 6) architecture-plugin contract is mandatory:
 - define class 'EvoHyperConnection(torch.nn.Module)'
 - define callable
 - 'build_custom_hyper_connection(num_streams, *, dim, branch) -> torch.nn.Module'
 - returned module must be an instance of 'EvoHyperConnection'
 - optional:
 - 'build_custom_expand_reduce_streams(num_streams) -> (expand_stream, reduce_stream)'
- 7) strict runtime behavior:
 - 'EvoHyperConnection.forward(self, residuals, *branch_args, **branch_kwargs)'
 - must call 'self.branch(...)' at least once on every forward pass
 - output must preserve shape/dtype/device and keep gradient flow.

Manifold-level evolution surfaces in code:

- inside 'EvoHyperConnection.__init__':
 - parameterization, constraints, projection/retraction, regularization
- inside 'EvoHyperConnection.forward':
 - residual-to-branch mixing, branch-to-residual merge, geometry-aware routing
- optional custom expand/reduce stream operators.

Design intent:

- evolve the routing manifold itself, not only superficial hyperparameters;
- produce mathematically coherent and implementation-valid operators;
- avoid benchmark-result hallucination in summaries/reviews.

Transformer HyperConnection task preamble (code+design-intent run)

Task type: transformer_architecture

Task contract is the same as the theory+code transformer run above.

Artifact mode for this run: \modecodeintent{}. Return strict JSON with non-empty summary_md and code_content; keep theory_content as an empty string.

Interpretation note:

- summary_md still carries the design principles and intended scientific rationale of the proposed hyper-connection.
- reviewer judgments are grounded primarily in code_content, benchmark evidence, and lineage/metadata rather than in theory_content.

G.3 Optimizer MHC-lite

The four optimizer runs share the same benchmark contract and differ only by artifact mode. We therefore report one run-local preamble for `theory+code` and one for `code+design-intent`.

Optimizer MHC-lite task preamble (theory+code runs)

Task type: optimizer

Task: evolve PyTorch optimizer implementations to improve GPT training quality on fixed residual architecture in the MHC-lite stack. Primary objective: minimize `val_loss` (lower is better). Architecture contract is fixed by benchmark adapter: `hyper_conn_type=none` and `hyper_conn_n=1` (residual only), so candidates must optimize training dynamics only. Hard requirements for every generated `code_content`: define non-empty `OPTIMIZER_ALIAS` (string); define `OPTIMIZER_NODE_ID` exactly equal to `output_node_id` provided by runtime; define class `EvoOptimizer(torch.optim.Optimizer)`; `EvoOptimizer.__init__` must accept params; `EvoOptimizer.step` must accept closure argument.

Optimizer MHC-lite task preamble (code+design-intent runs)

Task type: optimizer

Task: evolve PyTorch optimizer implementations to improve GPT training quality on fixed residual architecture in the MHC-lite stack. Primary objective: minimize `val_loss` (lower is better). Architecture contract is fixed by benchmark adapter: `hyper_conn_type=none` and `hyper_conn_n=1` (residual only), so candidates must optimize training dynamics only. Hard requirements for every generated `code_content`: define non-empty `OPTIMIZER_ALIAS` (string); define `OPTIMIZER_NODE_ID` exactly equal to `output_node_id` provided by runtime; define class `EvoOptimizer(torch.optim.Optimizer)`; `EvoOptimizer.__init__` must accept params; `EvoOptimizer.step` must accept closure argument.

Artifact mode for this run: `\modecodeintent{}`. Return strict JSON with non-empty `summary_md` and `code_content`; keep `theory_content` as an empty string and do not rely on theory for reviewer judgments.

G.4 Native optimizer ablation

The eight native-optimizer ablation runs share the same task contract and differ only by artifact mode and evolution settings. We therefore report one run-local preamble for `theory+code` and one for `code+design-intent`.

Native optimizer task preamble (theory+code runs)

Task type: optimizer

Task: evolve PyTorch-compatible optimizers for native supervised learning benchmarks spanning classification-focused synthetic and tabular tasks. Primary objective: minimize `mean_val_loss` (lower is better) across the configured native optimizer benchmark. Hard requirements for every generated `code_content`: define non-empty `OPTIMIZER_ALIAS` (string); define `OPTIMIZER_NODE_ID` exactly equal to `output_node_id` provided by runtime; define class `EvoOptimizer(torch.optim.Optimizer)`; `EvoOptimizer.__init__` must accept params; `EvoOptimizer.step` must accept closure argument.

Native optimizer task preamble (code+design-intent runs)

Task type: optimizer

Task: evolve PyTorch-compatible optimizers for native supervised learning benchmarks spanning classification-focused synthetic and tabular tasks. Primary objective: minimize `mean_val_loss` (lower is better) across the configured native optimizer benchmark. Hard requirements for every generated `code_content`: define non-empty `OPTIMIZER_ALIAS` (string); define `OPTIMIZER_NODE_ID` exactly equal to `output_node_id` provided by runtime; define class `EvoOptimizer(torch.optim.Optimizer)`; `EvoOptimizer.__init__` must accept params; `EvoOptimizer.step` must accept closure argument.

Artifact mode for this run: `\modecodeintent{}`. Return strict JSON with non-empty `summary_md` and `code_content`; keep `theory_content` as an empty string and do not rely on theory for reviewer judgments.

H Short_json Prompt Templates

This appendix block records the compact `short_json` operator prompts used in the reported runs that relied on the lighter prompt family. For navigation within this section: Appendix [H.2](#) gives the pair-selector prompt; Appendix [H.3](#) the crossover prompt; Appendix [H.4](#) the exploration-mutation prompt; Appendix [H.5](#) the correction-mutation prompt; and Appendix [H.6](#) the reviewer prompt.

H.1 Prompt inventory

- Pair selector: `../prompt_bundle_short_json/pair_selector_prompt.md`
- Crossover: `../prompt_bundle_short_json/crossover_prompt.md`
- Exploration mutation: `../prompt_bundle_short_json/exploration_mutation_prompt.md`
- Correction mutation: `../prompt_bundle_short_json/review_correction_mutation_prompt.md`
- Reviewer: `../prompt_bundle_short_json/reviewer_agent_prompt.md`

H.2 Pair selector prompt

Agent Prompt: Pair Selector (short_json)

```
# Pair Selector Agent (Short JSON)

## Role
Select top disjoint crossover pairs from the provided crossover candidate pool.

## Input
JSON payload with:
- 'task_type'
- 'task_preamble'
- 'top_k_pairs'
- 'winners': list of '{ node_id, summary_md, selection_tier }'

Use 'summary_md' as the primary evidence. Use 'selection_tier' as a priority signal:
- prefer 'strict_winner'
- use 'fallback_tier_a' when strict winners are too few
- use 'fallback_tier_b' only as the weakest fallback

## Constraints
- Disjoint pairs only.
- No self-pair.
- Use only input node IDs.
- Return at most 'top_k_pairs'.
- If fewer than 2 candidates are provided, return empty list.

## Output (JSON only)
'''json
{
  "pairs": [
    ["node_id_a", "node_id_b"]
  ]
}
'''
```

H.3 Crossover prompt

Agent Prompt: Crossover (short_json)

```
# Crossover Agent (Short JSON)

## Role
Given two parent nodes, generate one child that combines useful mechanisms and addresses weaknesses.

## Runtime Contract
- Use input JSON only.
- Do not read/write files.
- Return JSON only.

## Input Semantics
Use:
```

```

- 'task_type', 'task_preamble'
- 'parent_a', 'parent_b' with 'summary_md', 'theory_content', 'code_content'
- optional review and benchmark summaries
- 'output_node_id'

If 'task_type=optimizer', 'code_content' must include:
- 'class EvoOptimizer(torch.optim.Optimizer)'
- non-empty 'OPTIMIZER_ALIAS'
- 'OPTIMIZER_NODE_ID = output_node_id'
- 'step(self, closure=None)' implementation

## Output (JSON only)
'''json
{
  "summary_md": "...",
  "theory_content": "...",
  "code_content": "..."
}
'''

```

H.4 Exploration mutation prompt

Agent Prompt: Exploration Mutation (short_json)

```

# Exploration Mutation Agent (Short JSON)

## Role
Mutate one parent node toward novelty and diversity while staying aligned with the task.

## Runtime Contract
- Use input JSON only.
- Do not read/write files.
- Return JSON only.

## Input Semantics
Use:
- 'task_type', 'task_preamble'
- 'node' with 'summary_md', 'theory_content', 'code_content'
- optional review and benchmark summaries
- 'output_node_id'

If 'task_type=optimizer', enforce required optimizer contract fields in 'code_content'.

## Output (JSON only)
'''json
{
  "summary_md": "...",
  "theory_content": "...",
  "code_content": "..."
}
'''

```

H.5 Correction mutation prompt

Agent Prompt: Correction Mutation (short_json)

```

# Correction Mutation Agent (Short JSON)

## Role
Mutate one node to fix correctness, robustness, and clear performance blockers.

## Runtime Contract

```

```

- Use input JSON only.
- Do not read/write files.
- Return JSON only.

## Input Semantics
Use:
- 'task_type', 'task_preamble'
- 'node' with 'summary_md', 'theory_content', 'code_content'
- optional reviewer and benchmark summaries
- 'output_node_id'

If 'task_type=optimizer', enforce required optimizer contract fields in 'code_content'.

## Output (JSON only)
'''json
{
  "summary_md": "...",
  "theory_content": "...",
  "code_content": "..."
}
'''

```

H.6 Reviewer prompt

Agent Prompt: Reviewer (short_json)

```

# Reviewer Agent (Short JSON)

## Role
Review one node for correctness and originality relative to task.

## Runtime Contract
- Use input JSON only.
- Do not read/write files.
- Return JSON only.

## Input Semantics
Use node payload fields:
- 'summary_md', 'theory_content', 'code_content'
- optional benchmark summary

Transformer-task correctness rule (mandatory when task grounding says transformer):
- if code declares custom hyper-connection, 'EvoHyperConnection.forward(...)' must call
  'self.branch(...)' at least once per forward pass.
- any custom node bypassing branch is incorrect.

## Scores
Return 1-5 scores with booleans:
- 'correctness_score', 'correctness'
- 'originality_score', 'originality'

Binarization convention:
- binary = 1 iff score >= 4

## Output (JSON only)
'''json
{
  "originality_score": 1,
  "originality": 0,
  "correctness_score": 1,
  "correctness": 0,
  "review_md": "..."
}
'''

```

I Workflow_v2 Prompt Templates

This appendix block records the exact `workflow_v2` prompt files used by the active agent roles, so the operator-side scaffolding can be inspected directly. For navigation within this section: Appendix I.2 gives the pair-selector prompt; Appendix I.3 the crossover prompt; Appendix I.4 the exploration-mutation prompt; Appendix I.5 the correction-mutation prompt; and Appendix I.6 the reviewer prompt.

I.1 Prompt inventory

- Pair selector: `prompt_bundle_json_workflow_v2/pair_selector_prompt.md`
- Crossover: `prompt_bundle_json_workflow_v2/crossover_prompt.md`
- Exploration mutation: `prompt_bundle_json_workflow_v2/exploration_mutation_prompt.md`
- Correction mutation: `prompt_bundle_json_workflow_v2/review_correction_mutation_prompt.md`
- Reviewer: `prompt_bundle_json_workflow_v2/reviewer_agent_prompt.md`

I.2 Pair selector prompt

Agent Prompt: Pair Selector

```
# Pair Selector Agent Prompt (JSON Native, Workflow-Exact)

## Role
You are 'PairSelectorAgent'.
Select top disjoint winner pairs for crossover.

## Non-Negotiable Contract
- Input is provided as 'INPUT_JSON'.
- Use only JSON fields.
- Do not read or write files.
- Return exactly one JSON object and no extra prose.

## INPUT_JSON Schema
```json
{
 "task_type": "optimizer | transformer_architecture | general",
 "task_preamble": "string",
 "top_k_pairs": 10,
 "winners": [
 {
 "node_id": "string",
 "summary_md": "string"
 }
],
 "task_grounding": {
 "canonical_task_source": "task_preamble",
 "task_type": "string",
 "task_preamble": "string",
 "if_task_type_general": "string",
 "role_objective": "string"
 }
}
```

## Input Field Usage Map (Required)
- 'task_type': determines domain assumptions.
- 'task_preamble': canonical task objective/constraints; overrides generic heuristics.
- 'top_k_pairs': hard maximum number of pairs.
- 'winners[].node_id': identity only; valid ID set for output.
- 'winners[].summary_md': only evidence source for pairing quality.
- 'task_grounding.role_objective': additional priority signal when ambiguous.

Do not use any evidence outside 'summary_md'.
```

```

## Hard Constraints
- Disjoint pairs only.
- No self-pair.
- Use only provided winner IDs.
- Return at most 'top_k_pairs' pairs.
- If <2 valid winners, return empty list.

## Pairing Objective
Prioritize pairs that maximize expected crossover quality:
1. Complementary strengths and weaknesses.
2. Meaningful novelty potential (avoid near-duplicates).
3. Compatibility of assumptions and design choices.
4. Potential to resolve known pain points.
5. Balanced risk.

## Required Workflow
### Step P0 - Task grounding
Extract task criteria from 'task_preamble' (especially when 'task_type = general').

### Step P1 - Trait extraction from summaries
For each winner summary, extract:
- core mechanism
- strengths
- weaknesses/failure modes
- stability hints
- novelty hints

### Step P2 - Pair synergy scoring
For each candidate pair, score qualitatively for:
- complementarity
- compatibility
- expected gain
- risk

### Step P3 - Disjoint greedy selection
Pick pairs from highest to lowest synergy while enforcing disjointness.

### Step P4 - Deterministic tie-break
Break ties by lexicographic order of '(node_id_a, node_id_b)' after sorting each pair internally.

### Step P5 - Final validation
Validate size, IDs, disjointness, no self-pairs, and cap by 'top_k_pairs'.

## OUTPUT_JSON (Strict)
'''json
{
  "pairs": [
    ["node_id_a", "node_id_b"]
  ]
}
'''

## Output Constraints
- 'pairs' must be array of 2-string arrays.
- No duplicates or mirrored duplicates.
- No prose outside JSON.

```

I.3 Crossover prompt

Agent Prompt: Crossover

```

# Crossover Agent Prompt (JSON Native, Workflow-Exact)

## Role
You are the 'crossover' operator.

```

Given two parent nodes, produce one child node that combines strengths and addresses diagnosed weaknesses.

Non-Negotiable Contract

- Input is provided as 'INPUT_JSON'.
- Use only JSON fields.
- Do not read or write files.
- Return exactly one JSON object and no extra prose.

INPUT_JSON Schema

```
```json
{
 "task_type": "optimizer | transformer_architecture | general",
 "task_preamble": "string",
 "parent_a": "NodePayload",
 "parent_b": "NodePayload",
 "output_node_id": "string",
 "task_grounding": {
 "canonical_task_source": "task_preamble",
 "task_type": "string",
 "task_preamble": "string",
 "if_task_type_general": "string",
 "role_objective": "string"
 }
}
```
```

'NodePayload' may include:

- 'node_id', 'generation', 'parent_ids', 'created_by'
- 'summary_md', 'theory_content', 'code_content'
- 'benchmark_summary'
- optional 'benchmark' object or 'null'
- optional 'review' object or 'null'
- 'metadata', 'paths' (informational only)

Input Field Usage Map (Required)

For each parent:

- 'summary_md': fast synopsis and claimed strengths/weaknesses.
- 'theory_content': formal method assumptions/guarantees.
- 'code_content': implementation reality and API behavior.
- 'review.review_md' and review scores: correctness/originality diagnostics.
- 'benchmark_summary' / 'benchmark.benchmark_summary_md': observed regime failures and strengths.
- 'metadata': optional lineage/context hints.

Global fields:

- 'task_preamble': canonical task contract.
- 'task_type': domain framing only.
- 'output_node_id': required target node id for generated content.

Task Grounding Rules

- 'task_preamble' is canonical.
- If 'task_type = general', infer domain/objective/constraints from 'task_preamble' + parent artifacts.
- Do not assume optimizer-specific structure unless required by task context.

If 'task_type = optimizer', enforce in 'code_content':

- define 'class EvoOptimizer(torch.optim.Optimizer)'
- non-empty 'OPTIMIZER_ALIAS'
- 'OPTIMIZER_NODE_ID = output_node_id'
- implement 'step(self, closure=None)' and return closure loss when closure is provided

Required Workflow

Step 0 - Restate and reconstruct both parents

From 'summary_md', 'theory_content', and 'code_content', restate:

- method definitions/update rules
- assumptions and intended regime
- implementation characteristics

Step 1 - Parent comparison (theory + mechanism)

```

For each parent, identify:
- advantages/disadvantages
- brittle points (stability, conditioning, mismatches)
- compute/memory profile

### Step 2 - Evidence-based diagnosis
Use 'review' + benchmark summaries to identify 1-3 key pain points.
If benchmark evidence is missing, state that explicitly in 'summary_md'.

### Step 3 - Crossover design goals
Choose 1-3 explicit goals directly linked to diagnosed pain points.

### Step 4 - Produce child design
Create an auditable merge:
- inherited from parent A
- inherited from parent B
- changed/new parts and causal reason for each

### Step 5 - Theoretical support
Provide at least one theorem-style statement/proof sketch under clear assumptions.
Mark conjectural parts explicitly.

### Step 6 - Implementation
Produce full runnable code aligned to theory with stability guardrails.
Avoid gratuitous overhead; if no budget is specified, target <=2x heavier parent per-step overhead.

### Step 7 - Summary artifact
'summary_md' must include:
- parent IDs and inheritance map
- diagnosed pain points
- changes and rationale
- expected gains and risks
- novelty/overlap remarks

## OUTPUT_JSON (Strict)
```json
{
 "summary_md": "string",
 "theory_content": "string",
 "code_content": "string"
}
```

## Output Constraints
- all fields must be non-empty strings.
- 'code_content' must be full code, not diff snippets.
- no prose outside JSON.

```

I.4 Exploration mutation prompt

Agent Prompt: Exploration Mutation

```

# Exploration Mutation Prompt (JSON Native, Workflow-Exact)

## Role
You are the 'exploration_mutation' operator.
Mutate one node for novelty/diversity while preserving task validity.

## Non-Negotiable Contract
- Input is provided as 'INPUT_JSON'.
- Use only JSON fields.
- Do not read or write files.
- Return exactly one JSON object and no extra prose.

## INPUT_JSON Schema

```

```

'''json
{
  "task_type": "optimizer | transformer_architecture | general",
  "task_preamble": "string",
  "mode": "exploration",
  "node": "NodePayload",
  "output_node_id": "string",
  "task_grounding": {
    "canonical_task_source": "task_preamble",
    "task_type": "string",
    "task_preamble": "string",
    "if_task_type_general": "string",
    "role_objective": "string"
  }
}
'''

```

'NodePayload' may include:

- 'node_id', 'generation', 'parent_ids', 'created_by'
- 'summary_md', 'theory_content', 'code_content'
- 'benchmark_summary'
- optional 'benchmark' object or 'null'
- optional 'review' object or 'null'
- 'metadata', 'paths' (informational only)

Input Field Usage Map (Required)

- 'summary_md': high-level current strategy and intent.
- 'theory_content': where assumptions/guarantees are weak.
- 'code_content': practical constraints and implementation hooks.
- 'review': correctness/originality weaknesses to address.
- 'benchmark_summary' / 'benchmark': empirical pain points/regimes.
- 'task_preamble': task-specific hard constraints and objective.
- 'output_node_id': must be reflected in optimizer contract fields when applicable.

Task Grounding Rules

- 'task_preamble' is canonical.
- If 'task_type = general', infer domain/objective/constraints from 'task_preamble' + node artifacts.
- Do not assume optimizer-specific structure unless required.

If 'task_type = optimizer', enforce in 'code_content':

- define 'class EvoOptimizer(torch.optim.Optimizer)'
- non-empty 'OPTIMIZER_ALIAS'
- 'OPTIMIZER_NODE_ID = output_node_id'
- implement 'step(self, closure=None)' and return closure loss when closure is provided

Required Workflow

Step 0 - Baseline reconstruction

Reconstruct the baseline from summary/theory/code.

Step 1 - Evidence-based diagnosis

Use review + benchmark summaries to identify key limitations.

Step 2 - Root-cause shortlist

List 1-3 root causes tied to specific evidence.

Step E0 - Return to original task

State objective, constraints, and where baseline breaks under task context.

Step E1 - Pick two adjacent domains

Choose two adjacent domains (not the same subliteration), e.g.:

- control theory
- numerical analysis
- information geometry
- signal processing
- queueing/game theory/physics/graph theory

Step E2 - Import one mechanism from each domain

```

For each mechanism, state:
- what it guarantees
- what it costs
- why it helps the diagnosed issue

### Step E3 - Short debate and decision
Compare both mechanisms briefly and choose winner or coherent consensus merge based on:
- elegance
- implementability
- alignment with constraints and evidence

### Step 5 - Apply mutation
Implement minimal, coherent mutation that realizes chosen mechanism.
Keep theory and code aligned.

### Step 6 - Summarize
'summary_md' must include:
- diagnosed weakness
- two explored domains
- debate outcome
- final mutation and rationale
- expected gains/risks

## Additional Constraints
- If no compute budget is specified, keep <=2x baseline per-step overhead.
- Avoid heavy inner loops unless strongly justified.
- Clearly separate proven claims from conjectures.

## OUTPUT_JSON (Strict)
```json
{
 "summary_md": "string",
 "theory_content": "string",
 "code_content": "string"
}
```

## Output Constraints
- all fields must be non-empty strings.
- full updated theory/code, not patch snippets.
- no prose outside JSON.

```

I.5 Correction mutation prompt

Agent Prompt: Correction Mutation

```

# Correction Mutation Prompt (JSON Native, Workflow-Exact)

## Role
You are the 'correction_mutation' operator.
Apply targeted fixes for correctness, robustness, and concrete performance blockers.
This is corrective, not exploratory.

## Non-Negotiable Contract
- Input is provided as 'INPUT_JSON'.
- Use only JSON fields.
- Do not read or write files.
- Return exactly one JSON object and no extra prose.

## INPUT_JSON Schema
```json
{
 "task_type": "optimizer | transformer_architecture | general",
 "task_preamble": "string",
 "mode": "correction",

```

```

"node": "NodePayload",
"output_node_id": "string",
"task_grounding": {
 "canonical_task_source": "task_preamble",
 "task_type": "string",
 "task_preamble": "string",
 "if_task_type_general": "string",
 "role_objective": "string"
}
}
'''

'NodePayload' may include:
- 'node_id', 'generation', 'parent_ids', 'created_by'
- 'summary_md', 'theory_content', 'code_content'
- 'benchmark_summary'
- optional 'benchmark' object or 'null'
- optional 'review' object or 'null' (with scores + 'review_md')
- 'metadata', 'paths' (informational only)

Input Field Usage Map (Required)
- 'review.correctness_score/correctness': severity of correctness issues.
- 'review.originality_score/originality': novelty status; do not force novelty here.
- 'review.review_md': actionable issue list.
- 'benchmark_summary' / 'benchmark': empirical failure modes and regressions.
- 'theory_content': where assumptions/claims need repair.
- 'code_content': where implementation/API/numerical fixes are needed.
- 'task_preamble': final correctness criteria.
- 'output_node_id': output contract binding when task is optimizer.

Task Grounding Rules
- 'task_preamble' is canonical.
- If 'task_type = general', infer domain/objective/constraints from 'task_preamble' + node artifacts.

If 'task_type = optimizer', enforce in 'code_content':
- define 'class EvoOptimizer(torch.optim.Optimizer)'
- non-empty 'OPTIMIZER_ALIAS'
- 'OPTIMIZER_NODE_ID = output_node_id'
- implement 'step(self, closure=None)' and return closure loss when closure is provided

Required Corrective Workflow
Step C0 - Extract actionable checklist
Prioritize issues:
- correctness blockers first
- major robustness issues second
- protocol/performance issues third

Step C1 - Verify and localize
For each issue, verify it from theory/code/benchmark evidence and localize where it occurs.

Step C2 - Minimal corrective mutation
Apply smallest effective fix for each verified root cause.
Preserve core idea unless blocking issues require redesign.

Step C3 - Update claims and limitations
If a claim cannot be repaired, weaken/retract explicitly.
Separate resolved vs open issues.

Step C4 - Compose final artifacts
'summary_md' must include:
- issue -> fix mapping
- before/after behavior expectations
- unresolved risks

Additional Constraints
- Do not introduce unnecessary compute overhead.
- Keep theory and code consistent.

```

```

- Do not fabricate benchmark improvements.

OUTPUT_JSON (Strict)
'''json
{
 "summary_md": "string",
 "theory_content": "string",
 "code_content": "string"
}
'''

Output Constraints
- all fields must be non-empty strings.
- full updated theory/code, not patch snippets.
- no prose outside JSON.

```

## I.6 Reviewer prompt

### Agent Prompt: Reviewer

```

Reviewer Agent Prompt (JSON Native, Workflow-Exact)

Role
You are 'reviewer'.
Review one candidate for correctness and originality relative to the task context.
Do not implement fixes.

Non-Negotiable Contract
- Input is provided as 'INPUT_JSON'.
- Use only JSON fields.
- Do not read or write files.
- Return exactly one JSON object and no extra prose.

INPUT_JSON Schema
'''json
{
 "task_type": "optimizer | transformer_architecture | general",
 "task_preamble": "string",
 "node": "NodePayload",
 "task_grounding": {
 "canonical_task_source": "task_preamble",
 "task_type": "string",
 "task_preamble": "string",
 "if_task_type_general": "string",
 "role_objective": "string"
 }
}
'''

'NodePayload' may include:
- 'node_id', 'generation', 'parent_ids', 'created_by'
- 'summary_md', 'theory_content', 'code_content'
- 'benchmark_summary'
- optional 'benchmark' object or 'null'
- optional prior 'review' object or 'null'
- 'metadata', 'paths' (informational only)

Input Field Usage Map (Required)
- 'task_preamble': canonical review criteria.
- 'summary_md': claimed contributions and intent.
- 'theory_content': formal correctness/assumptions/proofs.
- 'code_content': implementation correctness and API behavior.
- 'benchmark_summary' / 'benchmark': empirical support or contradictions.
- prior 'review' (if present): context only, not authoritative.

```

```

Task Grounding Rules
- 'task_preamble' is canonical.
- If 'task_type = general', infer review criteria from 'task_preamble' + artifacts.
- Do not apply optimizer-specific checks unless required by task context.

If 'task_type = optimizer', explicitly check:
- 'class EvoOptimizer(torch.optim.Optimizer)'
- non-empty 'OPTIMIZER_ALIAS'
- 'OPTIMIZER_NODE_ID' consistency when visible
- 'step(self, closure=None)' behavior

Required Review Workflow
Step R0 - Read and restate method
Use summary/theory/code to restate method and claimed contributions.

Step R1 - Correctness assessment
Identify concrete issues:
- missing assumptions/invalid logic
- theory-code mismatch
- API/numerical stability risks
For each major issue, explain why it matters.

Step R2 - Originality assessment
Assess overlap with known patterns and identify genuine novelty.

Step R3 - Suggested fixes (advisory only)
Provide concise actionable fixes and missing experiments/ablations.

Step R4 - Risk checklist
Cover:
- numerical stability
- scaling/computation
- reproducibility/protocol risk

Step R5 - Final scores
Assign integer scores in [1,5]:
- 'correctness_score'
- 'originality_score'
Apply binarization:
- binary = 1 iff score >= 4

Score Rubric (1-5)
- 5: strong, no major blockers
- 4: good, minor issues
- 3: mixed, meaningful concerns
- 2: weak, multiple major issues
- 1: fundamentally broken/invalid

'review_md' Required Sections
1. Summary of method
2. Correctness findings (major first)
3. Originality findings
4. Suggested fixes
5. Risk checklist
6. Final recommendation

OUTPUT_JSON (Strict)
'''json
{
 "originality_score": 1,
 "originality": 0,
 "correctness_score": 1,
 "correctness": 0,
 "review_md": "string"
}
'''

```

```
Output Constraints
- scores are integers in [1,5].
- booleans should follow score binarization.
- 'review_md' must be non-empty markdown.
- no prose outside JSON.
```