Latent-Cache Programs for Branching Reinforcement Learning

A higher categorical extension of Octopus with a Lean compiler for cloud training

Anonymous review draft

Abstract

Octopus trains a shared language-model backbone over forked, rewound, and cross-mounted key–value histories. We formalize these histories as typed causal diagrams with immutable state and define a smooth semantics for their numerical computations. Local solutions descend along covers of causal histories, and dependency-preserving packing commutes with evaluation and reverse differentiation. A streaming attention law gives the same exact-real output and pullback as packed attention. A versioned cache semantics separates address sharing from model compatibility and determines when eviction, migration, and heterogeneous mounts preserve a program. For learning, we prove the ownership-mask score identity and distinguish the episode gradient, the normalized clipped surrogate, and causal branch-removal credit. We derive bounds for recurrent interference, adaptive branch growth, and distributed execution. An accompanying Lean 4 compiler lowers a typed smooth lambda calculus and branch traces to a tensor plan. A PyTorch worker executes the plan, records replay scores, and performs grouped reinforcement learning on a reference hybrid model. The manuscript specifies matched-budget experiments for extending the original empirical results.

Keywords. Causal sheaves, reverse automatic differentiation, persistent KV caches, policy gradients, tensor compilation.

Contents

1 Introduction and relation to Octopus **3**

2 Persistent traces and cursor semantics **4**
2.1 Five control operations 4
2.2 Visibility, addressing, and canonicalization 6

3 Causal descent and compositional smooth semantics **6**
3.1 A topos of causal histories 6
3.2 Open diagrams and interfaces 8
3.3 Schedules and causal coherence 8
3.4 A smooth lambda calculus on each trace 9

4 Layer-selective mounting and packed execution **9**
4.1 Native positions and minimal continuation . . . 9
4.2 A quantitative separation of retrieval and recurrent state . . . 10
4.3 Rectangular attention and packed node layout . . . 11
4.4 Streaming attention without copying mounted K/V . . . 12

5 Adjoints of branched programs **13**
5.1 Interoperable projections . . . 14
5.2 Sensitivity of policy to intervention . . . 14

6 Likelihood, ownership, and grouped policy updates **15**
6.1 Leave-one-episode-out baselines . . . 15
6.2 The normalized clipped surrogate . . . 16
6.3 A smooth clipping family . . . 17

7 Causal branch credit **17**

8 Adaptive branching and resource bounds **18**
8.1 Expected depth, total work, and tails . . . 19
8.2 Pins, admission, and immutable memory . . . 20

9 Cloud semantics for replay, cache migration, and training **21**
9.1 Replay contract and eviction recovery . . . 21
9.2 Distributed reverse accumulation . . . 22
9.3 Tiering, locality, and experimental scaling . . . 22

10 Empirical extensions and evaluation protocol **23**
10.1 Deep-search performance and evidence conversion . . . 23
10.2 Context, cache reuse, and step normalization . . . 24
10.3 Cross-branch communication and policy drift . . . 25
10.4 Branch scaling, topology, and emergent roles . . . 25
10.5 Extending to a distributed training run . . . 26

11 A Lean compiler and cloud training worker **26**
11.1 Typing and lowering . . . 26
11.2 Trace plan and serving path . . . 27
11.3 Training and deployment boundary . . . 28
11.4 Executable checks and scope . . . 29

12 Conclusion **29**

A Coverage of the source report **30**

B A worked compiler witness **31**

C Architecture and deployment contracts **32**

1 Introduction and relation to Octopus

A language model that branches mid-episode acquires two kinds of memory. Its private history determines the causal evolution of its recurrent state and the ordinary keys available to its attention layers. Its borrowed history exposes other branches' keys and values without consuming their tokens as private input. The distinction survives backward propagation. A read can contribute a cotangent to a producer's key projection even though the producer's token appears exactly once in the probability of the executed episode. Treating the borrowed keys as another run of the producer's text changes both the dynamics and the learning objective.

Octopus [1] implements this distinction on a shared backbone. The full report describes immutable radix cursors, selective attention-layer mounting under native rotary positions, prefix-deduplicated training, and a clipped grouped policy update. It reports experiments in deep search, context utilization, attention interference, and branch scaling. The arXiv landing abstract additionally describes leave-one-branch-out marginal rewards and gives aggregate throughput and memory figures. The full report's displayed objective uses leave-one-*episode*-out group advantages. These two operations answer different questions. Sections 6 and 7 develop both, using the displayed full-report equations as the baseline for mathematical comparisons. The citation gives the PDF byline and records the landing-page metadata difference.

Our central object is a finite causal program whose edges carry typed data and whose events have stable identities. A worker reads a finite ordered collection of immutable key–value slices, including slices created elsewhere. A rewind changes the active private cursor. The earlier events still exist in the causal log, can have consumers in other branches, and can carry credit. Executing the program has two levels. A controller samples actions and chooses the next event from a discrete state. Given the events and token IDs, a differentiable evaluator computes a finite tensor map. This separation permits a smooth lambda calculus without differentiating the selection of a branch, and permits an exact likelihood for an executed stochastic topology without replacing its discrete choices by continuous relaxations.

The extension has four linked parts. First, a persistent transition semantics separates causal chronology, the private prefix tree, and the dataflow edges required for training. Its solutions descend over lower ideals of the causal order, and interface substitution is compositional. Second, a compilation theorem establishes equality of packed and reference evaluations and their pullbacks on the smooth stratum. We prove a streaming attention rule that avoids a materialized concatenation of cache blocks and show how to sum consumer cotangents at the unique producer. Third, we derive the episode score, characterize the effect of tree-size normalization, and give causal counterfactual and exact group baselines. Fourth, we specify a cloud protocol for reconstructing caches and publishing optimizer states, with a Lean compiler and a small hybrid tensor runtime that realize the executable fragment.

An extension of an empirical claim needs new data. We extend the original comparison designs into estimands and a matched-budget protocol. We give proofs where the question concerns semantics, resource counts, or learning identities. The source's search accuracies and hardware timings retain their status as reported observations. This distinction determines which rows of the coverage map in Appendix A can be discharged by proof and which require a research-scale experiment.

Conventions. Every dependency is oriented from producer to consumer. The event order is chronological and the tensor graph is acyclic. Parallel edges have distinct roles even when they connect the same pair of vertices. Token positions are zero based and slices are half open. A generated action has a stable occurrence identifier. A token value, the physical address holding its projection, its rotary position, and its action identifier are four different things. A key imported twice may contribute two occurrences to a softmax even when both occurrences refer to the same physical bytes.

2 Persistent traces and cursor semantics

Definition 2.1 (Episode event log)**.** An episode starts at a root and is recorded as a finite ordered log. An event has a unique ID, an episode-local branch ID, a type, an authorization context, and input handles pointing to earlier immutable snapshots. We distinguish an *append event*, which has a nonempty sequence of private tokens and a private parent, from a *control event*, which changes cursor visibility, branch state, or synchronization. Each private token carries its provenance and, when sampled, the decision identity, legal support, and behavior log-probability. A sealed step publishes stable ranges of its owned tokens. Token intervals exposed by a step never implicitly include intervals it borrowed.

For an append node u, write $p(u)$ for its private parent, x_u for the token vector of length q_u, M_u for the ordered list of mounted slices visible on that node, and s_u for the first native rotary position. The private edges form a rooted forest with a virtual empty trunk. Rewinding and then appending creates a new child of an earlier node. It does not add a backwards private edge. If a later reader borrows from an abandoned suffix, that suffix still occurs in the graph. The log also contains administrative events whose outputs affect what a later step is permitted to consume. The tensor graph is the dependency projection of this richer log, retaining private and borrowed dataflow.

A cursor consists of a branch identity, a private parent handle, the next private position, a visible mount list, a budget ledger, and the snapshot handles needed to restart computation. Per-layer checkpoint handles include recurrent hidden state, short convolution tails where present, and any other mutable numerical state required to continue. Parameters and cache provenance are immutable within a snapshot. The host records a branch-local stack of checkpoint capabilities. A fork can share the parent's private history without allowing a child to rewind the parent. A checkpoint restored in the child includes the child's own ledger and its private mount list at the time it was saved.

2.1 Five control operations

1. `fork` takes a live cursor and a finite list of child descriptors. It creates a fresh cursor for each child, with the inherited private handle, visible mounts and recurrence snapshot, and a separate growth ledger. Child-specific instructions enter as subsequent owned tokens. A handle duplication costs constant metadata work per child when the prefix and state are persistent. Materializing a mutable recurrent clone costs its state size.

2. `read_thread` resolves a published step range to an ordered list of typed slices. It validates that the publisher's immutable snapshot precedes the read and is authorized for the caller. The

reader gains a mount reference at selected attention layers and a position lower bound. Its current recurrent checkpoint is preserved. The transport can append a forced metadata token or sequence to make the new view available to the next sampling decision.

3. `wait` registers dependencies on a stated set of child completions and suspends the continuation until they are satisfied. A released wait may emit ordinary status tokens. It provides no latent communication on its own. The reference compiler consumes traces in which the completion dependencies of each recorded wait are already fulfilled.

4. `rewind` restores an authorized earlier checkpoint of the branch. It releases the active ownership of the intervening private suffix and restores the ledger and visibility at the checkpoint. It keeps the full execution log, including the action that requested rewind. A diagnostic note is a fresh append. External effects use independently specified idempotency keys or explicit compensation.

5. `done` seals the result, publishes the permitted snapshots, and marks the branch terminal. We use this full-report name for the close operation. The landing abstract calls the corresponding operation `commit`. A close event can be a deterministic interpretation of a sampled action or an external administrative step.

A log records the distinction between a sampled stop and an injected stop prompt. A forced termination prompt consumes an explicit reserve and carries no action score. If the model makes a subsequent sampled choice, that choice still has a probability. Similarly, a deterministic refusal to admit a branch and a scheduler sampling a branch to admit have different likelihood models. The ledger distinguishes private growth, imported visibility, physical bytes, and cumulative work. Rewind can release the first and perhaps the third. It does not undo the fourth. Keeping a separate monotone turn or work counter prevents unlimited rewind cycles under a nominally finite growth limit.

Proposition 2.2 (Causality and restoration)**.** *Assume every read refers to an earlier sealed snapshot, every continuation refers to an earlier private state, and the compiler assigns new immutable IDs to all fresh outputs. The tensor graph is acyclic. If replay uses the same model version, private parent, ordered mount list, positions, deterministic state-transition conventions, and input tokens, a restored checkpoint has the same mathematical continuation as its saved checkpoint. This remains true when other live branches retain references to the discarded suffix.*

Proof. Assign each tensor node the index of its defining event. Every private and mount edge strictly increases that index. A directed cycle would give a strict increase from an index to itself. A rewind changes only the live cursor pointing into the graph. Starting from the saved checkpoint fixes the recurrence initial state and the projections that are visible before the next append. For an identical next append, all deterministic layer updates use the same arguments. Induction gives equality for every finite continuation. Independent aliases to the abandoned suffix reference different immutable values and have no write access to the restored checkpoint. □

There are two forms of progress. A *chronological* path can include the failed suffix and the rewind command. A *private ancestry* path for the revised node skips that suffix and starts from the checkpoint. A release from wait can follow many branches chronologically while inheriting

exactly one private recurrent state. This distinction explains why a training graph reconstructed solely from final branch paths can omit both policy actions and numerical dependencies.

2.2 Visibility, addressing, and canonicalization

An addressable cache is a map from a typed key to a replica or to a recipe. The typed key includes the model weights and adapters, the input tokens and controls, the layer, the representation contract, and the dependency keys that affect the computation. To use a digest as an address, treat the serialized descriptor as canonical and compare full descriptors on collision or assume an explicit cryptographic collision bound. A position alone is never an address. A logical slice stores a versioned owner and its half-open bounds in the owner's tensor. Physical aliases may differ in lifetime and placement. PagedAttention and SGLang provide precedents for this distinction between logical cache sequences and physical storage [5, 6].

There are three different ways to share. Fork shares inherited private handles. Borrowing shares the projection of selected positions at selected layers. A rematerialization optimization shares equal subcomputations only when their input diagrams and contracts are equal. The first two are part of the operational meaning. The third requires proof of purity and equivalent dependencies. Identical token strings on unrelated branches can have different hidden states, rotary positions, and visible mounts. In addition, two independent samplings can return the same token. Deduplicating numerical evaluation must leave those two decisions in the likelihood.

Example 2.3 (A retained detour)**.** Root o forks a and b. Branch a appends a detour d, publishes it, and rewinds to its start. Branch b has already read a slice of d and now appends r. Branch a appends a replacement e, while o waits and later reads r. The private paths to e and r share the trunk. The dataflow still includes $d \to r$. Removing d because it is absent from both active histories would change the available keys for r. Counting d again because b reads it would change the policy score. Its one owner and any number of readers are both represented in the same log.

Proposition 2.4 (Occurrence identities)**.** *Let D be the set of sampled decision occurrences of a sealed log. A replay that preserves the private cursor and legal sampling support of each decision induces a bijection of policy factors indexed by D, even if it changes device placement or splits numerical segments. Copying a log segment into two numerical views does not add a decision. Identifying independent decisions with equal token values removes a factor.*

Proof. The factor of an occurrence is the probability of its sampled value given the declared history and support. A compiler may choose a new location for its computation and still recover the same factor with the recorded identity. Its occurrences are identified by the original sampling events, which remain fixed under a replay. A physical view is a map from an occurrence to a numerical address and can be many-to-one or one-to-many without changing the original indexing set. □

3 Causal descent and compositional smooth semantics

3.1 A topos of causal histories

Let P be the reachability poset of the full causal graph. It can include event dependencies irrelevant to a particular tensor pass. A downset $U \subseteq P$ is an available history closed under causal

prerequisites. Covering families of a history are collections of downsets whose union is that history. The basic open at v is $\downarrow v$. For fixed inputs, a local assignment on U attaches to each vertex in U a value of its type. When $v \in U$, it must satisfy the equation

$$z_v = f_v\big((z_{\sigma(e)})_{e:\tau(e)=v}\big).$$

Here σ and τ denote source and target and the edge tuple is ordered by port. Boundary vertices have specified or free inputs according to the interface. In this notation a borrowed value is an edge carrying a projection of $z_{\sigma(e)}$. The borrowing operation itself is a structural selection of that port.

Theorem 3.1 (Causal descent)**.** *The presheaf of local solutions on the downsets of P is a sheaf. More generally there is an equivalence*

$$\mathsf{Shv}(P_{\mathrm{Alex}}; \mathcal{S}) \simeq \mathrm{Fun}(P^{\mathrm{op}}, \mathcal{S}).$$

The space-valued extension of a diagram F on principal opens sends U to $\lim_{v \in U^{\mathrm{op}}} F(v)$. In particular, gluing can keep coherent identifications of checkpoint realizations.

Proof. For set-valued solutions, take compatible assignments on the members of a cover. The value at v is defined by any member containing v and is independent of the member by compatibility on the overlap. That member contains all predecessors of v, so the local defining equation holds there and also in the glued assignment. Uniqueness follows vertex by vertex.

For the space-valued assertion, in $\mathrm{Fun}(P^{\mathrm{op}}, \mathcal{S})$ let 1_U be the subterminal presheaf with support U. Pointwise, 1_U is the colimit of the representables $y(v)$ over $v \in U$. At an element $w \in U$, the category of $v \in U$ above w has initial object w, and at an element outside U it is empty. It follows that

$$\mathrm{Map}(1_U, F) \simeq \lim_{v \in U^{\mathrm{op}}} F(v).$$

A union cover is a pointwise effective epimorphism $\coprod_i 1_{U_i} \to 1_U$ with intersections as the summands of its Čech nerve. Mapping that colimit into F gives descent. On $\downarrow v$, the formula recovers $F(v)$ by Yoneda. Conversely the principal opens cover every U. A sheaf is determined by its values and restrictions on this basis, since intersections of basis opens are themselves covered by basis opens. These constructions are inverse. □

Taking the solutions as sets keeps exact assignments. We use the ordinary space-valued sheaf conventions of higher topos theory [18]. A space-valued version can instead record realizations of local equivalences and paths between them. For instance, place a space X_v of candidate realizations at a vertex and a space Y_e of interface states on each edge. The two ends map to Y_e. A global solution in spaces is the homotopy limit of the resulting incidence diagram, with specified input fibers when required. It retains paths witnessing agreement, including higher compatibility data. A causal history cover glues this data by the same basis argument. The resulting homotopy limits express interface coherence. Numerical reproducibility is a separate property of the realization map.

For parameters in an open $O \subseteq \mathbb{R}^d$, local functions $O \to X_v$ give a sheaf in the parameter coordinate as well. On rectangles $U \times O$, a family of smooth solutions can be glued in the causal coordinate by Theorem 3.1 and in the parameter coordinate by ordinary locality of smoothness. These two gluing operations commute because both are formed by limits. This describes internal

smooth fields of histories in a product site. The coefficients of one training run belong to a specific global section over its stratum of fixed discrete controls. Changing the topology or the action IDs moves to a different component of the program space.

3.2 Open diagrams and interfaces

An open numerical diagram has ordered input ports X and output ports Y. Its private and borrowed dependencies both appear as input edges of typed gates. A smooth gate may consume its parameters as extra inputs, and persistent reads may fan out one output into several gates. Composition glues matching output ports of the first graph to inputs of the second, preserving type and order. The input boundary has no incoming computation edges and gluing is directed from the first interior to the second. This orientation guarantees acyclicity. Parallel composition is disjoint union of interiors and concatenation of interfaces. Reassociation of finite gluings gives canonical isomorphisms, so these diagrams can be composed up to coherent renaming in the manner of open graph constructions [17].

On deterministic diagrams with one definition per output, evaluation is the map $f : X \to Y$ obtained by solving the equations in dependency order. It can be interpreted as the span $X \xleftarrow{\mathrm{id}} X \xrightarrow{f} Y$. Composition of these particular spans works in any category with the indicated pullback, which is the pullback along an identity and needs no transversality. General interfaces with auxiliary witnesses can be interpreted as spans in spaces, where composition uses homotopy pullback. If one uses both a forward value and a proof that a cache replica realizes it, the witness becomes part of the middle object and is carried across composition.

Theorem 3.2 (Interface substitution)**.** *Evaluation sends directed open-graph composition to composition of functions and parallel composition to products. Replacing a subdiagram by a diagram with the same smooth boundary map preserves the output and derivative of any directed smooth context into which it is substituted. For solution spaces with retained witnesses, gluing along an interface is modeled by the homotopy pullback of the two interface maps.*

Proof. Given an input value, each gate of the first diagram obtains the value it would have in isolation. The result at the second input boundary is exactly the value exported by the first. Induction on the second graph then gives the composite map. Disjoint diagrams have disjoint gate equations given their respective inputs, so their result is the product. Equality of smooth boundary maps implies equality after contextual composition and equality of their derivatives. In the space-valued incidence presentation, specifying a solution on the glued diagram is specifying both local solutions together with a coherent agreement on the identified boundary. The universal property of this space is the homotopy pullback. □

3.3 Schedules and causal coherence

A numerical schedule is a linear extension of the data dependencies. A controller schedule may have additional dependencies from authorization, sampling, state updates, and waits. For controller steps to commute, their resource footprints must also commute. In particular, a shared stateful random-number generator or a mutable shared tool response can impose an order even when two numerical nodes are incomparable. We use independently keyed draws in the schedule-invariance argument for executions with random choices.

Lemma 3.3 (Schedule exchanges)**.** *Any two linear extensions of a finite partial order are related by a sequence of swaps of adjacent incomparable elements. If the two computations at every such exchange read only their prerequisites and write disjoint immutable output IDs, all schedules evaluate the same assignments.*

Proof. Move the first element of the target schedule leftward in the other schedule. Any element crossed is incomparable with it. A prerequisite could not follow it in the target schedule, and a dependent could not precede it in the current schedule. After this move, remove the shared first element and use induction. For the evaluation assertion, swapping two eligible computations leaves both input tuples fixed. The pair of newly assigned values and all subsequent arguments are unchanged. □

This argument defines a functor from the free schedule-exchange groupoid to the groupoid of exact output assignments with equality as arrows. Each swap maps to an identity on the common assignment. Hence different exchange paths agree under the numerical interpretation, including paths built from commuting swaps on disjoint pairs. With schedules interpreted as device programs, a 2-cell can carry an explicit comparison of their emitted traces and tolerances. A numerical equality theorem does not imply that all device executions have the same costs or bit patterns.

3.4 A smooth lambda calculus on each trace

The simply typed language has finite tensor base types, products, arrows, and a signature of smooth tensor primitives. Variables denote projections. Abstraction and application are interpreted using exponential and evaluation maps in the cartesian closed category of diffeological spaces [19, 20]. A plot in a functional diffeology is smooth when its joint evaluation on arguments is smooth. This permits higher-order notation while keeping a fixed closed first-order tensor program an ordinary smooth map between finite-dimensional real spaces. One can form families of programs by varying smooth parameters in this semantics.

The controller uses a discrete tagged sum of trace shapes. Within a shape, selection of ports, discrete positions, masks, and token indices is fixed. Softmax at nonempty rows, affine maps, and hyperbolic tangent are smooth on their real domains. A masked attention row represents a softmax on a specified nonempty list of keys, so its semantics uses no real value corresponding to negative infinity. Hard top-k, a switching architecture, and the exact clipped objective are interpreted on their smooth strata or as piecewise smooth operations. A differentiable backend must state its derivative convention on stratum boundaries. The score of a sampled trace will come from its probability, with a derivative taken in the numerical parameters, rather than from a derivative of the trace selection map.

4 Layer-selective mounting and packed execution

4.1 Native positions and minimal continuation

For a cursor, let p be its next private position and let J be the finite set of native positions in the newly visible mounted slices. The invariant is that all older private keys and all currently visible

mounted keys are strictly earlier than the next query. If the invariant holds before a mount, define

$$p' = \max(\{p\} \cup \{j + 1 : j \in J\}). \tag{1}$$

The maximum over a set containing p handles an empty mount. A node of length q uses $p', p' + 1, \ldots, p' + q - 1$. A chronological monotonicity assertion concerns fresh continuations along a private ancestry path. Rewinding and appending from a saved checkpoint can reuse positions across different paths.

Proposition 4.1 (Minimal causal continuation)**.** *Equation* (1) *gives the least admissible next private position. It preserves the strict position invariant along a private ancestry path and extends to independent position axes by coordinatewise maximum when the contract requires strict pastness on each such axis.*

Proof. For integer indices, $q > j$ is equivalent to $q \geq j + 1$. Admissibility also requires $q \geq p$, so the least bound is the maximum. All old private and previously visible keys are less than p by the invariant. After an append, adding its positive length restores the invariant for its private and mounted keys. The coordinatewise claim repeats the argument in the product order. □

Following the rotary encoding of RoFormer [7], a block with angle rate ω_a acts by $R_a(\omega_a p)$. For the usual real dot-product convention,

$$\langle R(q)u, R(k)v \rangle = \langle u, R(k - q)v \rangle.$$

Keeping each mounted key at k and evaluating a new query at q uses this formula directly. Different key occurrences can have equal native positions. Visibility is determined by the event and key occurrence, so equality of numerical indices causes no aliasing in a causal mask. It can still change the distribution of rotary features relative to a different positional scheme. A production adapter must supply its actual position axes, block conventions, window rules, and supported index range. For learned absolute embeddings or sequence-length-dependent rescaling, changing position can alter other input coordinates and must be covered by the equivalence contract.

4.2 A quantitative separation of retrieval and recurrent state

At the event that mounts keys, the reader's private recurrent state remains unchanged. At future steps, a deeper recurrent block may consume features produced by attention over the borrowed keys. We give a pathwise bound for this indirect route. Compare two executions with equal sampled private tokens and equal direct inputs below the first retrieval layer. Suppose a particular recurrent block uses $h^+ = F(h, a)$ and obeys

$$\|F(h, a) - F(\tilde{h}, \tilde{a})\| \leq \alpha \|h - \tilde{h}\| + \beta \|a - \tilde{a}\|$$

on the pair of visited trajectories. If the two initial states agree and attention-mediated input differences are d_t, then

$$\|h_T - \tilde{h}_T\| \leq \beta \sum_{t=1}^{T} \alpha^{T-t} d_t.$$

Induction from the displayed bound proves the formula. If $d_t \le D$ and $\alpha < 1$, it yields $\beta D(1 - \alpha^T)/(1 - \alpha)$. A lower block whose causal cone excludes the mounted projection has $d_t = 0$ on these equal-token trajectories. Once the model samples different tokens, the two private histories can diverge even at a lower block. The equal-token condition keeps the state result distinct from a policy-level claim.

One can bound the effect of moving a native rotary position. For normalized attention scores with dot-product scale $1/\sqrt{d}$, a shift Δ of a key gives

$$|\Delta s| \le \frac{\|u\|\,\|v\|}{\sqrt{d}} \max_a \min(2, |\omega_a \Delta|).$$

This follows from the operator norm of the difference of two plane rotations. If a row has fixed visible key occurrences, then

$$\|\operatorname{softmax}(s) - \operatorname{softmax}(\tilde{s})\|_1 \le \|s - \tilde{s}\|_\infty.$$

Indeed the derivative in direction z has entries $p_i(z_i - \sum_j p_j z_j)$, whose absolute sum is bounded by the square root of the variance of the random variable taking z_i with probability p_i. This is at most $\|z\|_\infty$. Integrating along the line segment proves the bound. With $\|v_i\| \le V$, the associated output error is at most $V\|s - \tilde{s}\|_\infty + \max_i \|v_i - \tilde{v}_i\|$. These estimates concern interventions with a fixed list of occurrences. Inserting or dropping keys changes the softmax denominator and requires an estimate on the added probability mass.

4.3 Rectangular attention and packed node layout

Split the log into nonempty token nodes at changes of private ancestry, mount visibility, or positional continuation. Let n_u denote the number of earlier key occurrences selected for node u at a particular layer, after applying its architecture's visibility contract. Put all n_u earlier occurrences first and place the q_u self keys last. At local query row i with $0 \le i < q_u$, the valid columns in a standard causal self block are

$$0 \le j \le n_u + i.$$

This is precisely the bottom-right causal triangle of a q_u by $(n_u + q_u)$ rectangle. It is valid because the prefix is visible to every row and the suffix contains self keys in local order. The positional jump ensures that cached keys advertised as past really are past for all those rows. A sliding window, grouped mask, or per-row mount change can violate the single-prefix premise. A split into smaller rows or an explicit visibility mask handles those cases. Kernel alignment conventions are part of the backend interface [10].

Write $U_1, \dots, U_m$ for a chosen topological enumeration. Store the node tensor outputs in one strip with offsets $b_i = \sum_{j<i} q_{U_j}$. Projection and pointwise layers operate on the strip without padding if they are independently tokenwise. At a recurrence layer, the boundary carries the predecessor's terminal checkpoint and convolution tails. At an attention layer, a slice descriptor selects the strip rows by producer and native position while the order of the strip has no positional meaning. Sparse experts can gather and scatter disjoint groups of rows conditional on recorded routing, provided that expert capacity and routing are invariant under the packing or implemented with the same dispatch decisions.

Theorem 4.2 (Packing commutes with evaluation)**.** *Assume fixed token IDs, deterministic numerical primitives, and stable occurrence identities. Assume each packed layer consumes the same ordered predecessor ports and uses the same per-row native positions and visibility as an occurrence-based reference evaluator. Assume recurrence boundaries carry all private causal state. Then unpacking the output of the packed execution gives the reference output at each node. If the active primitives are smooth on an open parameter neighborhood, their exact-real derivatives and pullbacks agree there as well.*

Proof. At the embedding layer the claim follows from the strip indexing. Suppose all values entering a layer agree. A pointwise or rowwise operation sees exactly its local arguments under the strip indexing. At recurrence, a topological induction over the nodes and an ordinary induction along their tokens give the same state transitions, since initialization is inherited along the private edge. At attention, each row uses the same query, key occurrences, positions, and values. The contract for rectangular attention gives the same active columns. Thus each layer agrees by induction. This establishes equality of the maps on the open neighborhood. Differentiation of equal smooth maps establishes the derivative and pullback assertions. □

The hypothesis about stable occurrences is stronger than a claim about unique memory addresses. If two producers give the same key vector and the reader intentionally includes both, the denominator counts both. Algebraic simplification can combine those occurrences with an explicit multiplicity weight, since their unnormalized masses add. Physical aliasing by itself leaves the occurrence list unchanged. A rewrite that removes duplicates from that list is valid only for a semantics that specifies a set or supplies the multiplicity correction.

4.4 Streaming attention without copying mounted K/V

For one query, partition the visible key occurrences into blocks. For a nonempty block B with scores s_i and value vectors v_i, define its stabilized statistics

$$m_B = \max_{i \in B} s_i, \qquad z_B = \sum_{i \in B} e^{s_i - m_B}, \qquad n_B = \sum_{i \in B} e^{s_i - m_B} v_i.$$

For two nonempty statistics define $m = \max(m_1, m_2)$ and merge them into

$$(m,\ e^{m_1 - m} z_1 + e^{m_2 - m} z_2,\ e^{m_1 - m} n_1 + e^{m_2 - m} n_2).$$

Treat an empty block as the identity, separately from this arithmetic formula. Query rows with different causal suffixes can mask the suffix independently while always including the earlier visible blocks.

Proposition 4.3 (Streaming output and pullback)**.** *For any partition of a nonempty visible row into finitely many blocks, the final ratio n/z of the merged statistics is the same smooth function of the active scores and values as dense attention. It therefore has the same pullback. Borrowed blocks can be supplied as read-only slice views.*

Proof. The merge rule computes the exact maximum of the union and converts both unnormalized sums into that common scale. By induction the result is precisely the statistic of the union,

independently of the reduction tree or block order in exact arithmetic. Cancelling the common exponent in the final ratio gives dense softmax attention. The ratio is smooth as a function of the original variables because the denominator is strictly positive. In automatic differentiation one can treat the stabilizing maximum as a detached scalar. For any fixed value of the scalar the numerator and denominator carry the same scale and their ratio is unchanged. This rule avoids assigning semantic significance to the arbitrary choice of a maximizing score at a tie. □

For a row with output $a = \sum_i p_i v_i$ and cotangent $\bar{a}$, the individual pullbacks are

$$\bar{v}_i = p_i \bar{a}, \qquad \bar{s}_i = p_i \langle \bar{a}, v_i - a \rangle.$$

A second block pass can evaluate these expressions using the final row statistics. It can deliver producer cotangents without allocating a contiguous copy of the borrowed keys or values. Score work and temporary row statistics still consume resources. One implementation can choose a streaming inference path and a packed gather for training. Their exact-real meanings agree under Theorem 4.2. Device precision, fused exponentials, and reduction order determine the numerical tolerance to use for replay comparisons. FlashAttention and FlashAttention-2 exploit closely related online-softmax identities for hardware efficiency [8, 9].

5 Adjoints of branched programs

Treat the value at node v as a finite-dimensional real vector z_v that includes any ports needed downstream. Let E_{uv} be the multiset of occurrences of edges from u to v. The local map is $f_v(z_{\mathrm{in}(v)}, \theta)$ and the scalar loss is $L(z, \theta)$. We may regard parameters as extra source nodes. A partial derivative $D_e f_v$ acts on the particular port labeled by e, including a slice projection if only a subrange of a producer is used.

Theorem 5.1 (Unique producers, additive cotangents)**.** *The reverse traversal*

$$\lambda_u = \partial_{z_u} L + \sum_v \sum_{e \in E_{uv}} (D_e f_v)^* \lambda_v, \qquad \nabla_\theta L = \partial_\theta L + \sum_v (D_\theta f_v)^* \lambda_v$$

gives the total derivative with respect to the parameters and, by the analogous source formula, the input cotangents. The traversal applies to private, borrowed, and rewind-induced dependencies.

Proof. An arbitrary perturbation satisfies $\delta z_v = \sum_{e:\tau(e)=v} D_e f_v \, \delta z_{\sigma(e)} + D_\theta f_v \, \delta\theta$. Starting from δL, eliminate each δz_v in reverse dependency order. The coefficient of each predecessor perturbation increases by the transpose applied to the current coefficient of δz_v. This accumulates every incidence exactly once, and the leftover parameter coefficient is the stated gradient. There is no separate reverse rule for historical logs. A rewind determines which earlier checkpoint is a private argument, and other retained edges contribute independently. □

Categorically, copying a value for k consumers uses the diagonal $x \mapsto (x, \ldots, x)$. Its derivative is the diagonal on tangent vectors and its reverse map sums the consumer covectors. This is the structural part of reverse differentiation in a cartesian reverse differential setting [21, 22, 23]. In particular, a borrowed slice with no new owned action score can still change the producer's

parameter gradient. Replacing its live edge by a constant with the same primal value generally changes the derivative.

For a LoRA key projection [11] $K = (W + BA)H$, a reader cotangent G_K contributes $G_K H^\top A^\top$ to B and $B^\top G_K H^\top$ to A when the token states are columns. The corresponding H cotangent $(W + BA)^\top G_K$ propagates upstream. Values and native rotations have their own transposes. A system that recomputes keys using current weights can compute these derivatives even when rollout saved only a detached behavior snapshot. A system that supplies old detached keys to the current model evaluates a different parameterized map.

Corollary 5.2 (Hessian-vector propagation)**.** *If the active maps and loss are twice continuously differentiable, a tangent pass followed by differentiation of the reverse recurrence computes a Hessian-vector product. For an edge contribution $(D_e f_v)^* \lambda_v$, the differentiated contribution is*

$$(D_e f_v)^* \delta\lambda_v + \big(D(D_e f_v)[\delta z_{\mathrm{in}(v)}, \delta\theta]\big)^* \lambda_v.$$

Thus a reusable packed linearization also supports higher derivatives on a fixed smooth stratum.

Proof. Apply the product rule to the backward equation of Theorem 5.1 and the ordinary chain rule to the forward perturbations. The reverse order is still acyclic and determines the perturbed cotangents uniquely. The perturbed parameter coefficient is the desired directional derivative of the gradient. □

5.1 Interoperable projections

For heterogeneous producer type P and reader type Q, a mount adapter is a typed smooth map $A_{P,Q} : K_P \times V_P \to K_Q \times V_Q$ together with a position and representation contract. Its incoming cotangent is obtained by the reverse chain rule in Theorem 5.1. Such an adapter can be used for cross-scale or role-specialized communication even when the private states of the two architectures have different types. A hardware alias is valid when the underlying representation is identical or related by a verified zero-copy view. A learned adapter generally computes a new tensor. If transitive mounts pass through successive representation changes, an optimizer may collapse the chain only when it implements the same composite adapter. Adapters with an identity and declared composition law form a typed diagram, so associative composition gives the same smooth boundary map and cotangent under either parenthesization.

5.2 Sensitivity of policy to intervention

Fix an observation protocol and compare two policies that differ only in an allowed intervention. For a bounded event H, the difference of its probabilities under episode laws P, Q is at most $\|P - Q\|_{\mathrm{TV}} \le \sqrt{\mathrm{KL}(P\|Q)/2}$ [24]. When all external conditional kernels agree and action laws are absolutely continuous, the chain rule expresses the episode KL as a sum of expected conditional action KLs up to the stopping time, with terminal padding by deterministic actions. A conditional probability bound from a short-horizon intervention study can therefore be lifted if a corresponding bound on the full sum is available. The action-level KL numbers in the source measure replay alignment under particular comparisons. They cannot be multiplied or reinterpreted as complete episode guarantees without the units, weighting, and number of decisions.

6 Likelihood, ownership, and grouped policy updates

Let $\theta \in O$ be an open real parameter domain. Fix the prompt, external kernel specification, action supports, and sampling protocol. We assume a finite horizon and finite action sets for the exact likelihood statements. The environment and forced controls have parameter-independent kernels conditional on the executed history. The policy probabilities on the allowed supports are positive and differentiable. The trace records the observation at each decision, including authorized latent reads and the placement of forced bridge tokens. Schedules may be deterministic or sampled by a declared parameter-independent kernel conditional on the realized history. A learned scheduler has its own policy factors.

Theorem 6.1 (The score of an owned trace)**.** *Let $\mathcal{A}(\tau)$ be the decision occurrences in a complete trace τ, with history H_a and chosen action x_a. For any trace in the common support,*

$$P_\theta(\tau) = C(\tau) \prod_{a \in \mathcal{A}(\tau)} \pi_\theta(x_a \mid H_a), \qquad S_\theta(\tau) = \nabla_\theta \log P_\theta(\tau) = \sum_{a \in \mathcal{A}(\tau)} \nabla_\theta \log \pi_\theta(x_a \mid H_a).$$

A flattened collection of trace views gives the same score if an ownership mask selects exactly one instance of each decision with its recorded history.

Proof. Factor the probability of the chronological log into conditional probabilities. Multiplying the parameter-independent factors gives $C(\tau)$. Every sampled private action contributes its conditional factor. Inherited tokens, imported K/V, forced tool messages, and fixed administrative decisions contribute no action factor of their own. The chain rule for the logarithm gives the score. The masked sum merely changes the indexing, because it contains each required summand once. □

The recorded discrete history is fixed when computing the derivative of each factor. Its numerical realization can depend on θ through the entire causal diagram. In particular the conditional logits of a reader depend on parameters through live borrowed projections. Theorem 5.1 differentiates this dependence. This explains why unique action scoring and repeated consumption of one numerical value are compatible. It also explains why a trajectory of terminal leaves alone is insufficient. A rewind decision removed from the active prefix is still a realized policy choice.

If the logged observable G forgets some decisions, then

$$\nabla_\theta \log P_\theta(G = g) = \mathbb{E}_\theta[S_\theta(\tau) \mid G = g]$$

on fibers of positive probability. Different logs can have the same observed graph. The identity follows by differentiating the finite sum over the fiber and dividing by its probability. A compiler that records only the resulting graph needs enough provenance to reconstruct these hidden factors or must use a marginalized likelihood.

6.1 Leave-one-episode-out baselines

For $m \geq 2$, take conditionally independent episodes τ_i sampled at the same prompt and policy. Let $R_i \in [-1, 1]$ be their rewards and let

$$B_i = \frac{1}{m-1} \sum_{j \neq i} R_j, \qquad A_i = R_i - B_i.$$

With binary success y_i, the source uses $R_i = 2y_i - 1$. Taking the episode cost as exogenous once the trace is fixed, the average of $A_i S_\theta(\tau_i)$ is unbiased for the gradient of the expected reward conditional on the prompt. Indeed $\mathbb{E}_\theta[S_\theta] = \nabla_\theta 1 = 0$ and B_i is independent of the sampled trace conditional on the prompt. RLOO uses the other group's episodes to construct a baseline without fitting a critic [14]. If the episodes share an external random seed that affects their laws, the argument applies after conditioning on a common context for which conditional independence and the score identity hold. Exchangeability alone is insufficient.

An action-specific baseline $b(H_a)$ is also valid under the local score identity, even when the number of decisions is random and later decisions depend on earlier ones. Pad a stopped process by deterministic actions up to the horizon and condition each summand on its pre-action history. The conditional expectation of the action score vanishes. The same argument permits a randomized baseline measurable with respect to a context independent of the current draw given its history. With delayed credit, a branch-wide baseline used for multiple decisions needs this measurability at every affected decision.

6.2 The normalized clipped surrogate

Write θ_0 for the behavior weights and $Z_i = |\mathcal{A}(\tau_i)| > 0$. For a sampled action a in episode i define $r_{ia} = \pi_\theta(x_a \mid H_a)/\pi_{\theta_0}(x_a \mid H_a)$. The source report's Eq. (9) has sample objective

$$J_{\mathrm{clip}}(\theta) = \frac{1}{m}\sum_i \frac{1}{Z_i} \sum_{a \in \mathcal{A}(\tau_i)} \min\{r_{ia}A_i, \mathrm{clip}(r_{ia}, 1-\epsilon, 1+\epsilon)A_i\}. \tag{2}$$

This is a per-action PPO-style surrogate using an episode advantage [15, 16]. Given a finite batch and $\epsilon > 0$, all ratios lie in the unclipped band on a neighborhood of θ_0. Thus its gradient at θ_0 is the batch average of $A_i S_{\theta_0}(\tau_i)/Z_i$. Removing the Z_i denominators gives the exact unweighted score estimator there. If every Z_i is a fixed common constant, the normalized update differs by that constant. Otherwise, the random denominator changes the score weighting.

Proposition 6.2 (Length-weighting identity)**.** *For a fixed prompt, independent baseline B with mean b, and nonzero random Z, the normalized score update at behavior weights satisfies*

$$\mathbb{E}\left[\frac{R-B}{Z} S_{\theta_0}\right] = \nabla_\theta \mathbb{E}_\theta[R/Z]|_{\theta_0} - b\, \nabla_\theta \mathbb{E}_\theta[1/Z]|_{\theta_0}\,.$$

The baseline mean on the right is evaluated under the behavior distribution.

Proof. Separate the reward and baseline terms. Conditional independence gives $\mathbb{E}[(B/Z)S_{\theta_0}] = b\,\mathbb{E}[S_{\theta_0}/Z]$. Differentiating the corresponding finite expectations gives the two terms. □

For a concrete instance, let a binary choice have probability $p = 1/(1+e^{-\theta})$. Its reward is the chosen bit and a second parameter-independent fair draw occurs only for bit one, so its decision count is one or two. With $p = 1/2$, the gradient of expected reward is $1/4$. With the leave-one-episode-out baseline, the expected normalized score is $3/16$. This follows from $S(0) = -1/2$, $S(1) = 1/2$, and $b = 1/2$. The two answers differ even at ratio one with no clipping in a neighborhood. This effect motivates measuring the update's dependence on trace size, particularly when failed large trees and successful small trees have different Z.

For exact off-policy expectation under common support and equal environment kernels, the importance ratio of a complete trace is

$$W(\tau) = \frac{P_\theta(\tau)}{P_{\theta_0}(\tau)} = \prod_{a \in \mathcal{A}(\tau)} r_a.$$

Summation over traces gives $\mathbb{E}_{\theta_0}[WR] = \mathbb{E}_\theta[R]$. The product can have high variance and does not follow from averaging individual clipped ratios. Replayed latent features should be recomputed under θ for the numerator. A comparison at θ_0 first checks equality of the behavior model, positions, mount lists, tokenization and masks. An intentional policy update to θ then gives ratios with their ordinary off-policy meaning.

6.3 A smooth clipping family

For $\tau > 0$, define $s_\tau(x) = \tau \log(1+e^{x/\tau})$ and the smooth minimum $m_\tau(x, y) = -\tau \log(e^{-x/\tau}+e^{-y/\tau})$. If $l = 1 - \epsilon$ and $u = 1 + \epsilon$, set

$$c_\tau(r) = l + s_\tau(r - l) - s_\tau(r - u).$$

Replacing the clipped gain by $m_\tau(rA, c_\tau(r)A)$ makes it smooth in the current policy weights wherever the action probabilities are smooth and the logged support is fixed. The value difference from the clipped gain is bounded by

$$\tau \log 2\,(1 + 2|A|).$$

In fact each softplus differs from the positive part by at most $\tau \log 2$, and the exact minimum is 1-Lipschitz in either argument with the other held fixed. The smoothing is useful for higher-order optimization on a fixed batch. Uniform value convergence alone gives no uniform convergence of derivatives across the clipping boundary. Hard clipping and smooth clipping are different objectives at positive temperature and can be compared with the same behavior traces.

7 Causal branch credit

A branch can be credited at three different levels. Its sampled actions have likelihood scores. Its K/V and checkpoints can receive numerical cotangents. Its presence can affect the reward of a re-executed episode. The first two are local to one executed trace, while the third is an intervention on a structural model. The landing abstract's branch-removal language fits the third question. A safe intervention must define how a removed branch changes capabilities, scheduling, and the continuation of its readers.

Let ξ collect the external random variables and keyed action draws. A policy and a scheduler give a structural function $\mathcal{R}_\theta(C, \xi)$, where C is a set of permitted branch capabilities. At each draw, a stable semantic key identifies the source of randomness. Shared keys may be coupled between interventions, and new decisions can use independent keys. Removing a branch suppresses its spawn and publication, then reruns all computations whose inputs or legal supports change. A reader of the removed branch can use a declared fallback. The reward difference for branch b is

$$\Delta_b(C) = \mathbb{E}_\xi[\mathcal{R}_\theta(C, \xi) - \mathcal{R}_\theta(C \setminus \{b\}, \xi)].$$

This is a total intervention effect under the specified fallback. One can instead remove a particular mount capability while leaving both branches active. That identifies a narrower interface effect under the chosen controls.

Simply deleting a tensor edge from a logged completed trace evaluates an altered fixed-token numerical program. It can measure replay sensitivity. It cannot simulate which tokens, waits, or tool queries would have been chosen without that edge. Replacing the reader's continuation by its counterfactual one is particularly important when the branch controls which evidence is available at the root.

Proposition 7.1 (Paired estimation and additive allocation)**.** *Let rewards lie in* $[-1, 1]$ *and let the random seeds be independent across* n *paired trials. Their mean reward difference is unbiased for* $\Delta_b(C)$ *and satisfies*

$$\mathbb{P}(|\widehat{\Delta}_b - \Delta_b| \geq t) \leq 2e^{-nt^2/8}.$$

For $v(C) = \mathbb{E}_\xi[\mathcal{R}_\theta(C, \xi)]$ *on a finite capability set* B*, averaging successive insertion effects over all permutations yields allocations* ϕ_b *with* $\sum_{b \in B} \phi_b = v(B) - v(\varnothing)$*.*

Proof. Each paired difference has the target expectation by linearity and lies in $[-2, 2]$. Hoeffding's inequality [25] gives the stated tail. For a permutation, the successive differences telescope from the empty set to B. Average the telescope over all permutations to obtain the allocation identity. This is the cooperative game value of Shapley [26]. □

For a fixed coupling, the variance of a paired difference is the sum of the two marginal variances minus twice their covariance. Common draws can reduce variance when paired successes are positively correlated. The coupling itself changes the variance of the estimator while preserving the mean when both marginals are correct. Query-level resampling should retain the pairs, especially when different queries have widely different baseline success.

Example 7.2 (Complementary branches)**.** Suppose either of two branches alone has reward zero and both together have reward one. Their two full-coalition removal effects are both one. They sum to two although the joint gain is one. The permutation allocation assigns one half to each branch. Redundant branches have the opposite behavior when either can independently solve the task. Branch removal effects remain interpretable as conditional intervention effects without an additive allocation premise.

A reward from a branch-removed rerun can serve as a score baseline for a decision if it is conditionally independent of that decision's draw given the decision history. For example, construct it by simulating an environment where the target decision is absent and the future exogenous noises are conditionally independent of its draw. Then the conditional-score proof applies. If the removed branch is selected because the original decision succeeded, or the intervention shares a random variable that determines the original decision, this proof does not apply. An empirical intervention can still be a meaningful attribution. Its use inside a gradient estimator is a separate assertion.

8 Adaptive branching and resource bounds

The original experiments distinguish explored tokens, surviving private history, active prompt size, and critical-path generated tokens. A shared-prefix program gives each of those a different

accounting rule. Consider a sealed episode with token-node lengths q_u. Its unique token work is $Q = \sum_u q_u$. A leaf-expanded representation uses $\sum_u m_u q_u$ private token occurrences, where m_u is the number of leaf paths through u in the chosen expanded private tree. Their difference is exactly $\sum_u (m_u - 1) q_u$. All dead-end paths carrying policy scores must be represented in the comparison. The ratio depends on where branching happens and on shared prefix lengths. It can be close to one for short prefixes and grow linearly with the number of leaves for long shared trunks.

At one attention layer, the number of visible score pairs in the rectangular contract is

$$\sum_u \left(q_u n_u + \frac{q_u(q_u + 1)}{2} \right).$$

It is possible to reduce Q while increasing this sum by adding mount visibility. Holding model shape and per-pair kernel work fixed, the count separates work due to executing fresh tokens from work due to retrieving evidence. The block-statistic rule can reduce materialized intermediate memory while the same score pairs are evaluated. This distinction matters when interpreting an increase in owned tokens processed per training step as a wall-clock throughput result.

Let W be the total cost of unit-time tasks in an execution DAG and S its longest directed path. With P identical processors, any schedule needs at least $\max(W/P, S)$ time. A greedy schedule has time at most $W/P + S$ in this model. In each step with fewer than P ready tasks, all ready tasks are executed, so the longest residual path decreases by at least one. There are at most S such steps. Fully occupied steps use P units of work and there are at most W/P of them. Communication, asynchronous retrieval, and heterogeneous devices require explicit tasks or additional constraints. The longest private chain is only a lower bound on S when waits or cross-branch dependencies increase the span.

8.1 Expected depth, total work, and tails

For generations of an adaptively spawned tree, let Z_d be the number of workers in generation d, and let $\mathcal{F}_d$ expose the information used to bound the next generation. The root may have a different reproduction law.

Theorem 8.1 (Conditional subcriticality)**.** *Suppose* $\mathbb{E}[Z_1] \le \lambda$ *and* $\mathbb{E}[Z_{d+1} \mid \mathcal{F}_d] \le \mu Z_d$ *for all* $d \ge 1$*, with* $0 \le \mu < 1$*. Then*

$$\mathbb{E}\left[\sum_{d \ge 1} Z_d \right] \le \frac{\lambda}{1 - \mu}, \qquad \mathbb{P}(\textit{a worker exists at depth } d) \le \lambda \mu^{d-1}.$$

The tree has finitely many workers almost surely. If each worker's conditional expected cumulative work is at most w *given its birth history, expected total worker work is at most* $w\lambda/(1 - \mu)$*.*

Proof. Iterate conditional expectation to get $\mathbb{E}[Z_d] \le \lambda \mu^{d-1}$. Apply monotone convergence to sum these bounds. Markov's inequality bounds $\mathbb{P}(Z_d \ge 1)$. Finite expectation of the nonnegative worker count implies almost-sure finiteness. For work, index potential workers by birth order, condition on the history of each worker that is born, and sum the conditional work bounds. □

Independence of siblings is unnecessary. The conditional bound is stronger than an observed average child count. As a counterexample to reasoning from a finite rollout average, sample a

hidden regime at the root that makes every subsequent worker produce two children with small positive probability and no children otherwise. An unconditional mixture rate for a specified potential worker can be less than one. Conditional on having observed the large regime and on its active generation, the mean is two. A finite run stopped by a budget also cannot establish the uncapped reproduction law without accounting for censoring.

A stronger condition gives a tail on the number of workers. In a breadth-first exposure process, assume the initial queue has size at most r, and the next exposed worker has $X \in [0, b]$ children with $\mathbb{E}[X \mid \mathcal{F}] \leq 1 - \delta$ for $\delta > 0$. For sufficiently small $t > 0$, there is a uniform $q < 1$ with

$$\mathbb{E}[e^{t(X-1)} \mid \mathcal{F}] \leq q.$$

This follows by Taylor expansion and the bounded second-order remainder, uniformly over histories. If Q_n is the remaining queue after n workers are examined and N is the total, the update on a nonempty queue is $Q_{n+1} = Q_n + X - 1$. Applying the conditional moment bound to the killed process gives $\mathbb{E}[e^{tQ_n}\mathbf{1}_{N>n}] \leq e^{tr}q^n$. Consequently $\mathbb{P}(N > n) \leq e^{tr}q^n$. An unbounded root count can be incorporated when its exponential moment is finite. This result isolates which stronger data a study needs to support tail guarantees.

The stop protocol also affects growth measurements. Active-slot limits and forced terminal prompts change the observed process. A comparison that measures unconstrained demand should record whether another spawn was attempted or would have been accepted under a larger limit. A comparison of deployed cost should include timeouts and hard budget terminations as realized events.

8.2 Pins, admission, and immutable memory

Suppose cache objects have sizes $b(k)$ for keys k. Let L be the set reachable from live cursors and pins, and R the budgeted bytes reserved for in-flight private work and transfers. The physical cache occupation attributable to L counts each resident object once. A safe admission invariant is

$$\sum_{k \in L_{\text{resident}}} b(k) + R \leq C$$

for device capacity C reserved for this pool. Represent reservations by typed capabilities with monotone consumption. Before accepting an event, atomically reserve its worst-case new bytes. When it installs a page, debit the reserved bytes and credit the resident object exactly once. A fork or read can add pins with zero new K/V data in a shared pool, while still reserving cursor and view metadata. Release removes a reference. Physical reclamation is legal only after the last pin and in-flight read are gone.

If fork creates k fresh handles it requires at least k handle writes. Prefix bytes need no copying. Private recurrent checkpoints can be referenced persistently, or materialized at a cost proportional to their size. These separate counts refine the constant-time terminology of the source. Reference-counting a root handle can be constant in prefix length if descendants are retained through reachability. Updating a count on every page is proportional to the number of pages. The two implementations have the same storage semantics with different operation costs.

9 Cloud semantics for replay, cache migration, and training

A cloud run separates a deterministic plan from resource placement. The compiler output identifies logical nodes, slices, and policy actions. A worker maps these to device addresses and may temporarily rematerialize a producer. The control plane holds immutable model and tokenizer versions, signed or otherwise authorized job descriptors, budgets, and the durable update pointer. The data plane executes the tensor graph. Cross-host communication transfers owned messages or uses an authorized remote view where supported. Device-local zero copying refers to reference sharing inside an address space. Transfer across placement domains has a transport cost even if no application-side intermediate copy is needed.

9.1 Replay contract and eviction recovery

A numerical descriptor contains the model and adapter digests, private parent descriptor, ordered mounted slice descriptors, architecture and positional conventions, numerical mode, and token delta. A behavior-log descriptor additionally contains the policy version, supports, action IDs, and external outputs. An optimizer checkpoint records the version it consumed, expected contributions, group reduction, optimizer hyperparameters, and the next version. A cache address is resolved only if the caller has authority over the descriptor's scope.

Proposition 9.1 (Coherent rematerialization)**.** *Suppose descriptors are complete for deterministic local evaluation. Rebuilding an evicted value from its descriptor and rebuilding any missing predecessor values in dependency order yields the original exact-real value. Migration to another device preserves the same assignment if decoding realizes the same primitive contracts. A change to a model or adapter version can reuse a cached object only under a verified equivalence for the affected subdiagram or when its full descriptor remains valid.*

Proof. Source descriptors determine the same inputs. Induct along the descriptor dependency order. At each step the existing or rebuilt predecessors have their original values, and the same primitive with the same representation arguments has a unique output. This proves reconstruction. A device realizing the same primitive maps is another implementation of the same induction. Changing a descriptor's numerical inputs changes the premise. An interface equivalence suffices by Theorem 3.2. □

A new LoRA snapshot can invalidate a cached numerical result even if the underlying text and position offsets agree. Keep behavior snapshots immutable while their episodes are active. If a newer trainer weights version recomputes those episodes, use the new version for every edge in the numerator and retain the recorded behavior probabilities for the denominator. This rule extends to persistent cross-episode latent memory. A memory bank can be modeled as an external input with declared historical representation, or as a recipe recomputed under current weights. These options induce different gradients. A parameterized adaptation map makes the choice explicit at the mount interface.

An incremental tokenizer or chat renderer has a related contract. Let D be the cursor descriptor and Δ the next turn. A valid incremental renderer produces $x(D) \parallel \mathrm{render}_{\Delta}(D, \Delta)$ equal to the canonical branch-preserving render, including termination markers and tokenization boundaries. This is an equality of token sequences. String equality of suffixes alone is insufficient if a tokenizer

may merge across the boundary. When the contract requires preserving earlier thoughts that ordinary chat rendering removes, the canonical render must encode that policy. A bounded startup probe supplies empirical compatibility evidence for a finite set of template patterns. A deterministic transducer with a known sufficient boundary state can give the universal property by induction on turns. A production adapter should record the renderer digest and reject mismatched replays.

9.2 Distributed reverse accumulation

Assign each differentiable edge occurrence a message ID made from the update epoch, episode, layer, producer slice, consumer and edge role. The receiver checks the expected set of incoming edges in the reverse graph. It reduces their contributions once per ID and seals the result once the expected set is complete. The empty expected set is a valid base case. Recurrent boundaries use the same mechanism as K/V slice boundaries. For tiled execution, a device may first combine its local messages as long as the manifest identifies the disjoint subset covered by the aggregate.

Theorem 9.2 (Distributed replay refinement)**.** *Suppose all admitted workers evaluate the same sealed descriptor and version, all required messages eventually arrive under retry, and a reducer counts exactly one well-typed contribution for every declared incidence. Over exact reals, placement, retries, and arrival order do not affect the computed gradient. If a finite-precision reproducibility target fixes the per-gate kernels, rounding modes, and reduction trees, every replay meeting that target has the same corresponding numerical evaluation.*

Proof. Forward correctness follows by induction from the sealed descriptors and Theorem 9.1. Reverse induction starts at the loss cotangents and applies the exact-once contribution rule at each producer. By Theorem 5.1, the set of all contributions, with multiplicity given by edge roles, is the reverse traversal of the declared diagram. Addition over exact reals is associative and commutative. Under the stronger finite-precision premise, each reduction and local operation is repeated in the same order with the same machine contracts. This proves the fixed numerical target. □

The optimizer step is a separate side effect. Suppose a durable store exposes a linearizable compare-and-swap on the head for one run. The updater stages an immutable checkpoint containing the new weights and optimizer state, the input version, and the complete contribution manifest. It then commits the new digest only if the old head still equals its input version. A fenced lease can restrict which updater is eligible to stage. A failed stage is unreachable from the head. Two valid updates of the same old head cannot both be successful because the first advances it. Retrying after an uncertain acknowledgement reads the head and checks the staged digest or the update ID. Progress requires an eventually available store and completion of the expected set. These premises isolate the exactly-once claim from the mechanics of any particular object store.

9.3 Tiering, locality, and experimental scaling

For a placement plan, model a missing object by alternatives with cost in the desired units. Rebuilding incurs compute and predecessor loads. Transferring incurs a fixed latency plus bytes divided by effective bandwidth. Retaining incurs capacity opportunity cost. This local estimate can choose a prefetch or materialization under a fixed layout. A global layout must consider shared ancestry, concurrent transfers, and slot limits. The invariant of the reference plan is independent

of that optimization. It is possible for multiple placement plans to refine the same causal program while their performance differs by orders of magnitude.

A cache that outlives an episode also needs lifetime and capability boundaries. Persistent metadata describes what can be borrowed, which model contract it realizes, and whether it can be regenerated after expiration. A new episode may import it as a sealed historical input. To train through that input's producer, include its generating descriptor and all required versions in the training graph. Treating a cached tensor as an exogenous fixed observation instead removes that gradient by definition. Neither a long retention period nor a large worker count follows from the semantic theorems. They become capacity and scheduling questions with measurable parameters.

10 Empirical extensions and evaluation protocol

The full report evaluates a Qwen3.5-4B backbone with sparse retrieval on BrowseComp-Plus and OpenResearcher [1, 2, 3]. It reports 49.6% BrowseComp-Plus accuracy for its recursive configuration against 32.1% for its 143K single-agent GRPO comparison. Its reported 51.2% OpenResearcher validation result is tied to the corresponding run and split in the report. Table 6 of the source lists more than one configuration and includes separate held-out, validation, and training figures. The two headline scores should therefore be referenced with their configuration, checkpoint, and benchmark denominator. Comparing a sparse BM25 pipeline with another system using a dense retrieval index is informative as an end-to-end comparison under those conditions. It does not estimate a causal effect of branching at fixed retrieval.

The full report provides empirical statements of four distinct kinds. The experiments below extend the question asked by each kind and specify what observations identify the extension. Analytic predictions give exact equality controls and resource identities. The research-scale experiments are proposed measurements.

10.1 Deep-search performance and evidence conversion

For search-trained policy baselines of the Search-R1 kind [4], use a frozen corpus snapshot and record its preprocessing digest, retriever configuration, search and open budgets, and retrieval tie-breaker. Use the same benchmark query IDs and grader for every system in a matched comparison. For each query, collect the returned answer, evidence IDs, generated tokens, inherited tokens, score-pair work, tool calls, wall-clock time, and estimated model FLOPs. Count truncated and timed-out episodes in the denominator. Apply evaluation only to held-out queries after choosing all configuration and budget settings.

Let Y indicate a correct final answer and E indicate at least one opened gold document. Report

$$\mathbb{P}(Y) = \mathbb{P}(Y \mid E)\mathbb{P}(E) + \mathbb{P}(Y \mid E^c)\mathbb{P}(E^c).$$

The source report's corresponding figures of approximately 0.589, 0.819, and 0.035 yield about 0.497 by this decomposition, consistent with its headline accuracy after rounding. This is a bookkeeping check on the interpretation of three different denominators. An intervention on evidence availability may also change the policy's search trajectory. To estimate the effect of enabling a retrieval tool or mount, rerun the policy with the intervention at the specified interface and pair by query.

Track the per-query fractions of annotated documents that are surfaced by search and opened by a tool, the event of opening any gold document, and exact answer correctness. Both fractional and at-least-one recall matter when one answer can be supported by several documents. A model may stop after opening one sufficient document. Conditioning on success and reporting only the fraction of correct answers with opened evidence changes the estimand. Report that fraction alongside the unconditional quantities. For unannotated but verified evidence, retain the document and grader ruling as a separate category.

Use three budget views. The first fixes a generated-token or model-FLOP budget. The second fixes the retrieval budget. The third fixes a deadline and device count. For a treatment T and baseline B, define

$$\widehat{\delta}_{T,B} = \frac{1}{n}\sum_{j=1}^{n}(\overline{Y}_{j,T} - \overline{Y}_{j,B}),$$

where bars average repeated seeds for the same query and budget. Bootstrap paired query units for query-level uncertainty, keeping all within-query runs together. If corpus subsets or independently trained checkpoints are sampled, report uncertainty at those higher levels as well. Estimate a cost–accuracy frontier by evaluating a prespecified grid of budgets. Avoid selecting the best run on the held-out split and presenting its result as if the choice were fixed.

10.2 Context, cache reuse, and step normalization

Instrument the same episode twice with packed and expanded evaluators at a common model version. In an equivalence comparison, each expanded reference row must see the same earlier key occurrences and checkpoints. An expanded reference that drops mount edges is a semantic ablation. The raw number of duplicated private token occurrences is determined by Section 8. The exact-real output and gradient equality prediction for equivalent evaluators is Theorem 4.2. Measure floating replay deviation and hardware time separately.

For each run, log the maximum active private window, maximum logical visible keys, total explored owned tokens, unique surviving tokens at termination, and unique bytes resident by tier. Measure the critical path on the full event DAG with durations for tool waits and transfers. Count a cache hit as a declared object resolved from the indicated tier without regeneration. State whether the denominator is objects, tokens, bytes, or lookups. Report wire bytes by control, token, cache payload, and gradient transfer. Separate warm and cold cache runs and report the behavior under eviction.

For training performance, record unique owned tokens per update, wall seconds per update, optimizer steps to reach a fixed held-out criterion, and peak allocated and reserved GPU memory. The source full report gives an increase in owned tokens per step for its packed training configuration. The landing abstract gives separate throughput and peak-cache ratios. Those metrics have different denominators. Using the same learning rule and hardware, compare packed and expanded traces at identical work limits. Then measure the system-level benefit of increasing the batch or tree size within the saved capacity. This distinguishes the compilation benefit from a changed learning regime.

Stratify the clipped objective by action count Z, failure mode, and topology. Compare unnormalized RLOO, a fixed normalization chosen from independent calibration data, the source's per-episode normalization, and positive-temperature smoothing. Record both reward and total

resources. Theorem 6.2 predicts a change in expected score weighting when Z depends on decisions. It does not predict which method will win a fixed-budget benchmark.

10.3 Cross-branch communication and policy drift

A paired read protocol should hold fixed the reader's pre-read trace and the producer snapshot. Vary the intervention among a blocked read, text replay, indiscriminate state injection where defined, unshifted mounts, relocated keys, and native-position selective mounts. Record source and reader lengths and rotary positions so that queries where a producer is ahead can be analyzed separately. For pure-attention, hybrid, and sparse-expert models, declare the selected layers and the full recurrent or router state contracts. Vary the selected layer set with the same positional convention to separate position handling from layer selection.

Report identifier exact match, semantic fact recall, negative log-likelihood on a frozen target, a task-deviation judge with documented calibration, and latency and memory. Let L_a denote expected negative log-likelihood for a shared evidence target under variant a. If $L_{\text{blocked}} > L_{\text{text}}$, one possible normalized retention measure is

$$\frac{L_{\text{blocked}} - L_{\text{mount}}}{L_{\text{blocked}} - L_{\text{text}}}.$$

Give the numerator and denominator as well as their ratio. The ratio can fall outside $[0, 1]$ and is unstable near a zero denominator. The full report describes its 92.2% figure as retention of full-text answer log-probability gain. A true conditional mutual information $I(F; S \mid H)$ for fact F, source S, and history H equals the expected log-density gain $\mathbb{E}[\log p(F \mid S, H) - \log p(F \mid H)]$ under the true joint law. A difference of model log losses estimates that information directly only under an appropriate calibration or model-correctness premise. The distinction gives a concrete extension of the original attention probe.

The source reports lower observed off-task behavior for its selective mount than for text injection. For a generalization test, rate task deviation blind to the read variant, report inter-rater reliability and paired uncertainty, and evaluate whether the estimate changes under different tool regimes. Replay the identical next tokens to isolate the immediate recurrence contract. Resample from the modified policy to estimate behavioral drift. The pathwise recurrence and episode-KL bounds of Sections 4 and 5 apply to different parts of this measurement.

10.4 Branch scaling, topology, and emergent roles

Vary maximum depth, worker budget, and test-time fork enablement independently of the trained weights. Compare zero-shot access to primitives, RL at a fixed shallow depth, zero-shot transfer to recursive branching, and training with recursive branching. Use paired seeds for toggling a control with fixed weights. Identify the intervention by the capabilities it changes and the fallback on an attempted forbidden spawn. Section 7 gives a matching construction for branch and mount effects.

For each training checkpoint, log first-turn delegation, worker share of search and open calls, root and worker rewinds, peer reads per edge type, and the complete depth histogram. Record proposed, accepted, and refused spawns, because a cap can censor the observed child count. Analyze topology and task accuracy jointly without selecting episodes solely on whether they complete naturally. Include a blinded phase analysis with a rule chosen in advance. For example, define

a change statistic as the contrast of means over two adjacent windows of fixed length and test its maximum over a declared checkpoint range using independently seeded training runs. This distinguishes a repeatable transition from a single conspicuous step in a plot.

For the guardrail study, treat loss masking, the answer reserve, and the retry prompt as separate interventions. Compare terminal rates, the fraction of termination actions owned by the policy, length and depth distributions, and rewards on the same task mix. Maintain a separate ledger for forced tokens and learned actions. For the source's architecture transfer question, evaluate a dense model, a gated recurrent-attention hybrid, a state-space hybrid, and a sparse-expert model with the same trace descriptors translated through their declared adapters. Exact cross-backend equivalence within each architecture is tested on fixed inputs. Behavioral equivalence between different model families has its own empirical target and need not hold.

10.5 Extending to a distributed training run

Run two configurations with the same logical episode graphs and optimizer schedule. First place each graph on one device and distribute only independent groups. Then split each graph along declared private and borrowed edges across two or more workers. Check identical manifest membership and compare ordered-reduction results. Measure inter-device bytes per edge role, transfer overlap, and sensitivity to placement. In a fault-injection run, repeat a delivered cotangent, terminate a worker before the update commit, and terminate it just after an acknowledged commit. The expected reference behavior follows from Theorem 9.2 and the version-pointer protocol. Test heterogeneous projections and persistent cache versions as separate treatments, because they may change the numerical model itself.

A minimal run manifest records the paper version, dataset and corpus digests, model and adapter versions, kernel backend, episode caps, hardware and network settings, RNG derivation, grader, and exact metric aggregation. Save the emitted plan, owned-action ledger, and behavior log-probability at each sampled action. Aggregate at the prompt, episode, and token levels with those weights named. All search and scaling claims can then be reproduced or extended under a declared comparison, while the companion's small CPU runs establish only the execution properties they directly exercise.

11 A Lean compiler and cloud training worker

The companion makes the separation between discrete control and smooth numerical evaluation executable. Lean 4.19 checks an episode and emits a canonical node plan. It also type-checks a smooth lambda term and lowers its first-order result to a tensor register program. A Torch interpreter runs these registers on CPU or CUDA tensors and uses the emitted trace for both incremental rollout and packed replay. The runtime's training objective can use either normalization in Section 6 and the paired rerun in Section 7. The research-scale architecture adapter remains a separate integration described below.

11.1 Typing and lowering

The kernel types are finite tensor shapes, products, and functions. A type $\mathsf{Ten}[d_1, \ldots, d_k]$ denotes a real array of the stated shape. An empty shape is a scalar. The compiler uses de Bruijn indices,

with zero referring to the most recently introduced binding. Its constructors include variables, abstraction, application, a sharing `let`, pairs, projections, elementwise addition and multiplication, matrix–vector multiplication, and hyperbolic tangent. Pointwise operations require identical source shapes, and matrix–vector multiplication checks both the rank and the inner dimension. A compiled kernel's external inputs and result have tensor types. First-class products and functions can be used internally and eliminated when lowering.

Lowering keeps an environment of register references, pairs, and closures. An abstraction packages its body with the current environment. Applying it evaluates the body with the argument in that captured environment. A `let` lowers its bound expression once and extends the environment with the result. Primitive applications allocate fresh registers after their operand registers, so every emitted operation refers to existing values. The implementation bounds normalization by fuel and returns an error when the bound is exceeded. It does not perform user-defined recursion.

Proposition 11.1 (Kernel lowering correctness)**.** *For a successful lowering of a well-typed term with first-order tensor inputs and output, interpreting the emitted registers over the declared real primitives gives the denotation of the input term. If these primitives are smooth, the two expressions have equal derivatives as functions of the input tensors.*

Proof. Relate an environment value of tensor type to the real value of its register. Relate an environment pair to the pair of its component denotations and a closure to the semantic function obtained by evaluating its body in its captured environment. The invariant is that the current register file contains correct values for all previously introduced tensor references. Induct on the successful lowering derivation, with closure application using the derivation of the body under the extended environment. Variables recover their denotations by the environment relation. Pairs and projections preserve the relation componentwise. Binding once and referencing its value realizes substitution by an equal denotation. Each primitive writes the value of the corresponding smooth map on registers that are already correct. The final tensor reference therefore denotes the term. Smooth equality gives derivative equality. □

The reference model's recurrent kernel is compiled from the closed-under-inputs expression

$$\mathsf{let}\ u = W_x x\ \mathsf{in}\ (\lambda v.\,\tanh(u+v))(W_h h).$$

Its register program evaluates two matrix–vector products, one pointwise sum, and one pointwise hyperbolic tangent. Projection and readout kernels are compiled in the same way. Torch implements the primitive maps and takes the reverse derivatives of the resulting tape. The Lean project checks two integer theorems about the native position jump, namely strict pastness and minimality. The graph, lowering, and derivative theorems in this paper are ordinary mathematical proofs. Correctness of Torch kernels and the Lean native runtime are trusted at the machine implementation boundary.

11.2 Trace plan and serving path

The compiler reads a JSON trace with explicit vocabulary, branch and position caps, an answer reserve, and a cumulative event cap. It checks fresh names, earlier published slices, legal private rewind targets, completion dependencies, action supports, and the association between sampled

actions and behavior log-probabilities. An append owns its tokens and can create a saved checkpoint. A read can include transitive mount occurrences in an explicitly chosen order. In the reference format, each completed append is publishable inside the same episode. An application with multiple tenants or finer visibility rules must perform the relevant capability checks before planning.

For autoregressive scoring, a sampled token in a node is scored by its predecessor output. The predecessor of a first token is the last token of its private parent. If a newly visible mount changes this predecessor view, the trace requires a forced bridge token before sampling. The bridge is a real input under the new visibility, and its output supplies the first post-mount distribution. Its own score is masked. This convention avoids scoring a sampled action with logits from a cursor that has not yet seen the read.

The incremental serving path extends an append-only resident store. A resident node contains its lower and upper recurrent terminal states, per-layer K/V pages, and logits. The store computes each new node once at a frozen behavior version and keeps old nodes through rewinds. The reader uses slices into the backing tensors and streaming attention. The packed replay path evaluates each unique node once per layer and computes the projections over the contiguous strip. It uses rectangular attention, a private recurrence pass, and gradient-preserving gathers. The trainer pairs these two paths by stable action occurrence. Both use two attention layers, a recurrent block below and above them, native rotary positions, and configurable low-rank attention projections. The default mount layer set contains only the first attention layer.

The toy environment is a bit-retrieval task. A scout receives a forced observation containing the bit. A reader may take a detour, request a rewind, and then mount the scout's hint. The root can sample whether to delegate, wait for its children, and synthesize an answer after a transitive read. Its final answer receives a binary reward. Sampled decisions choose within a finite recorded support. Administrative tool choices determined by those decisions are recorded separately. The paired audit removes the scout, removes the reader, or blocks the mounts and resamples the affected continuation under common keyed draws. This task supplies an end-to-end example of the original control and learning mechanisms in a tractable model.

11.3 Training and deployment boundary

Each worker collects complete groups at a fixed behavior version, retains action scores in the log, and performs one or more optimizer updates from those groups. The trainer recomputes live K/V under the current weights at each update. In its data-parallel mode, each rank draws a disjoint set of equally sized groups. It averages parameter gradients after local reverse evaluation. This realizes the mean of equally weighted rank objectives. It does not split one episode across hosts. The multi-host dependency protocol of Section 9 states the additional contract required for that use.

The worker can record trace manifests and atomically replace a checkpoint on a shared POSIX volume under a single-writer deployment. A restarted process checks the compiler, trainer, and training configuration digests before loading the optimizer and continuing with the next step. Draws are keyed by seed, step, rank, prompt group, and semantic decision label. Fixed kernels and hardware are additional conditions for bitwise reproducibility. The container definition and a Kubernetes GPU Job manifest provide a cloud launch path [27, 28]. The deployment expects an NVIDIA device plugin, a writable persistent volume, and an image hosted in the user's registry.

The reference code uses per-node tensor storage and a simple interpreter. Production page pooling, asynchronous tier migration, and optimized hybrid or expert kernels are adapter work.

An adapter for the source Qwen3.5, GDN, Mamba-2, or MoE systems must expose the complete private state, including convolution history where relevant. It must implement the tokenizer boundary contract, native position policy, group or window semantics, ordered K/V views, and immutable checkpoint extraction. Trainable expert routing needs either matching replay decisions or an explicit differentiable or piecewise-smooth contract. Appendix C gives the boundary checks used to determine when replacing the reference primitives preserves the theorem. The companion uses its own model and toy task. The source benchmarks, cross-family integration numbers, and GPU throughput require their corresponding models, datasets, and devices.

11.4 Executable checks and scope

The companion was built with Lean 4.19.0 and executed with PyTorch 2.7.1 on CPU. On the fixed witness in Appendix B, the float64 packed and incremental logit arrays differed by at most 5.56×10^{-16}. The largest absolute difference between packed and streaming parameter cotangents for the selected reader loss was 2.78×10^{-16}. The producer-only token's embedding gradient had norm 2.92×10^{-2} and was exactly zero in the fixed-input blocked-mount comparison. The mount's storage pointer was shared with its producer. The plan retained all three sampled actions and rejected future reads, invalid support, sampling before a required bridge, and rewinding to a foreign checkpoint.

The float32 training run used seed 17, rank-two low-rank attention projections, two groups of four episodes per step, and two optimizer updates per batch. Across two steps its largest behavior-to-replay action log-probability difference before the first update was 2.39×10^{-7}. The resumed checkpoint matched the uninterrupted model weights, optimizer state, and next step on the same CPU setup. A two-rank CPU launch reached initialization, where local socket restrictions prevented a collective run. The CUDA and Kubernetes paths are packaged for target-cluster validation. These measurements concern the compiled reference model.

12 Conclusion

A latent-cache program has an event log, a private ancestry relation, and a differentiable dataflow graph. The three views agree through stable occurrences and typed dependencies. The resulting semantics separates the effect of a cache optimization from a change to the model, and separates gradients through borrowed data from rewards attributed to branch interventions. The same separation explains the exact parts of packed DAG-GRPO and identifies where size weighting and clipping define a new objective. A cloud executor can preserve the program while changing its layout by carrying complete immutable descriptors and a declared replay contract. The reference compiler implements that interface for a smooth tensor fragment and an executable hybrid training task. The general theorems and evaluation protocol make the extension to the original scale a concrete program of measurement.

Authorship and computational evidence. This manuscript is prepared for anonymous review. The responsible author or group is to be confirmed in the editorial submission record.

An AI assistant contributed to drafting, code, and artifact preparation. The companion provides source, runnable compiler and training commands, and the verification record for the reference implementation. The reported Octopus results are attributed to the source paper.

A Coverage of the source report

The left and middle columns identify results in the full Octopus report and the additional landing-page claims. The last column states what the present paper supplies. Sections and tables in the first two columns refer to the source.

Source location	Original contribution or observation	Extension in this manuscript
Sections 1–2, Table 1, Fig. 1	Comparison of branching, long windows, context editing, stateless agents, and cache sharing	Typed criterion for distinguishing text replacement, state aliasing, and added dependency edges. Separate counts for live state, explored tokens, and work in Sections 2 and 8.
Section 3.1, Table 2, Fig. 2	Five primitives, shared prefixes, separate growth budgets, and rewind checkpoints	Immutable event model, causal restoration proof, explicit pin and reservation invariant, trace compiler and resident store.
Section 3.2, Eqs. 1–2, Figs. 3–4	Layer-selective mounts and native rotary positions	Minimal continuation proof, coordinatewise extension, recurrence influence and rotation perturbation bounds, segmented zero-copy views.
Section 3.3, Eqs. 3–4, Figs. 5–6	Incremental rendering and prefix-tree packed forward/backward	Renderer equivalence contract, layerwise packing theorem, rectangular mask characterization, streaming output and pullback, live gradient accumulation and Hessian-vector products.
Section 3.4, Eqs. 5–9, Fig. 7	Group leave-one-out advantages, owned loss masks, LoRA paths, and clipped tree objective	Trace score theorem, marginal likelihood, normalized-score identity and example, smooth clipping bound, joint importance ratio, compiled replay.
Section 4.1, Table 3, Fig. 8	Deep-search accuracy and compute comparisons	Matched-budget estimands with paired query-level analysis, clear retrieval and split controls, and auditable evidence and cost ledger.
Section 4.2, Tables 4–5, Fig. 9	Evidence conversion, prefix reuse, and communication versus drift	Conditional evidence decomposition, cache and wire denominators, normalized gain versus information, targeted replay and resampling protocol.
Section 4.3, Table 6, Fig. 10	Branch depth, role emergence, fork ablation, and packed scaling	Conditional growth and tail theorems, censored growth measurements, paired live interventions, checkpoint-level change analysis.
Section 5	Cross-model roles, persistent memory, and tiered multi-host scale	Typed mount adapters, snapshot version refinement, exact-once reverse contributions and optimizer commit contract.

Source location	Original contribution or observation	Extension in this manuscript
Appendix A	Template probes, reserves, termination, and admission	Canonical token-boundary contract, forced versus sampled ownership, cumulative work cap, guarded loss construction and measurement.
Appendix B, Table 7, Fig. 11	Recursive search example with retained detour and peer read	A minimal reproducible trace and distinct private and borrowed edge semantics. Appendix B and the compiler check.
Appendix C, Table 8, Fig. 12	Architecture-specific attention, MoE, GDN, and state-space hooks	Backend substitution contracts for routing, convolution tails, attention geometry and adjoints. Appendix C.
arXiv landing abstract	Leave-one-branch-out marginal rewards, aggregate throughput and memory figures	Causal intervention estimand, concentration and allocation, and explicit throughput and memory measurements with declared units.

B A worked compiler witness

In the included trace, root appends a two-token prompt at positions $0, 1$ and then forks workers a and b. Worker a appends four tokens at positions $2, 3, 4, 5$ and publishes them. Worker b appends its task at position 2 and samples an action in a subsequent detour. It then rewinds to its task checkpoint. Its current private parent is the task node and its next private position is 3. It mounts the half-open source interval $[1, 2)$, containing a's native key at 3, so Equation (1) advances the next position to 4. The forced bridge occupies that position and exposes its new attention view to the subsequent sampled report.

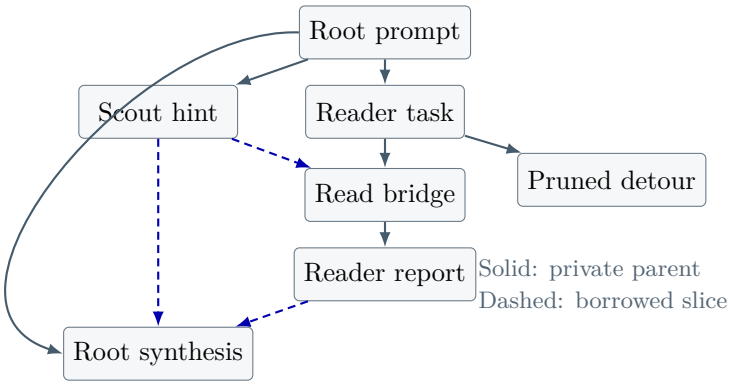


Figure 1: Private and borrowed dependencies in the reference witness. The rewind changes the reader's cursor back to its task. The detour retains its sampled action, while the bridge inherits state from the task and borrows a slice of the scout's hint.

When the root later waits and mounts the report, it requests the report's imported slice list as well as its own selected range. This produces two distinct mount occurrences in its synthesized read. The root still inherits its own private recurrence. In the packed plan the detour is a node even though its private suffix is absent from the final reader cursor. The three sampled actions

are the detour action, the report, and the answer. Each has one score. The published hint token contributes no likelihood factor because it was forced. Its embedding receives a nonzero reader gradient through the K/V edge. In the numerical blocked-mount comparison, the owned tokens and positions stay fixed and that isolated gradient is zero.

This trace represents the structural features of the original search walkthrough without reproducing its corpus or query. In a research-scale replay, each append would carry the real token delta and causal-convolution history. Each sealed tool output would have stable provenance, so a rerun can choose between reusing a historical answer for fixed-input comparison and resampling under an intervention.

C Architecture and deployment contracts

The original report describes seven supported model families and groups the integration into dense attention, sparse experts, gated recurrent-attention hybrids, and a state-space hybrid. The semantic adapter interface has five parts.

State shape and restore. A checkpoint is the complete state needed for a next token at the declared layer. Pure attention has no recurrent component. A short convolution needs its private last $w - 1$ inputs. A gated or state-space block needs its accumulated state and any additional normalization statistics. A persistent reference gives the same behavior as a clone when continuations use immutable values or copy on write. A backend test can compare a rewind continuation with direct execution from that checkpoint.

Projection and position. A projection declares layout, dtype, head grouping, position axes, rotary convention, and the masks under which a reader can consume a slice. A sliding or chunked attention layer must decide precisely which keys it sees under an index jump. A mixed architecture records the set of retrieval layers. If two readers use distinct position transforms, their shared native storage and the transforms applied at read time must be distinguished. An adapter that claims zero-copy aliasing also declares the memory domain in which that claim holds.

Dispatch and tiling. With tokenwise experts, a packed batch can group tokens by selected expert and scatter the output back to the strip. Capacity limits or load-coupled routing change the semantics unless the packed dispatch reproduces them. For fixed hard routing and smooth experts, derivatives are taken on that routing stratum. Activation tiling can cut memory use by checkpointing and recomputing numerical maps, with Theorem 9.1 supplying the exact-real argument. It must also preserve any keyed dropout decisions.

Backward ports. A mount interface supplies both the forward slice and a way for the slice cotangent to reach its producer. A recurrent state-space kernel supplies cotangents for the initial state and any convolution inputs. The reducer identifies each incidence even if two incidences point to the same physical range. The backend should compare both policy logits and producer-specific gradients on small sealed traces before using a faster fused path.

Control and reproducibility. Serving and training compare a versioned tokenizer and incremental renderer, an immutable weight snapshot, position tables, ordered visibility, and generated-action provenance. External tool call IDs are distinct from retry attempts. Checkpoint records distinguish a completed update from a partially collected group. These invariants apply to the source GDN and Mamba-2 designs [12, 13] and to other adapters with different internal computations.

References

[1] Tianming Sha, Junnan Li, and Siqi Zhao. *Octopus: One Brain, Many Branches for Latent-KV Multi-Agent Reinforcement Learning.* Technical report, arXiv:2611.06956v1, 2026. Full report and byline. Landing record and abstract. The landing record lists Tianming Sha.

[2] Zijian Chen et al. *BrowseComp-Plus: A More Fair and Transparent Evaluation Benchmark of Deep-Research Agent.* arXiv:2508.06600, 2025.

[3] Zhuofeng Li et al. *OpenResearcher: A Fully Open Pipeline for Long-Horizon Deep Research Trajectory Synthesis.* arXiv:2603.20278, 2026.

[4] Bowen Jin et al. *Search-R1: Training LLMs to Reason and Leverage Search Engines with Reinforcement Learning.* arXiv:2503.09516, 2025.

[5] Woosuk Kwon et al. *Efficient Memory Management for Large Language Model Serving with PagedAttention.* arXiv:2309.06180, 2023.

[6] Lianmin Zheng et al. *SGLang: Efficient Execution of Structured Language Model Programs.* arXiv:2312.07104, 2023.

[7] Jianlin Su et al. *RoFormer: Enhanced Transformer with Rotary Position Embedding.* arXiv:2104.09864, 2021.

[8] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness.* arXiv:2205.14135, 2022.

[9] Tri Dao. *FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning.* arXiv:2307.08691, 2023.

[10] Dao-AILab. FlashAttention reference implementation and causal-mask alignment notes. Repository documentation.

[11] Edward J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models.* arXiv:2106.09685, 2021.

[12] Songlin Yang, Jan Kautz, and Ali Hatamizadeh. *Gated Delta Networks: Improving Mamba2 with Delta Rule.* arXiv:2412.06464, 2024.

[13] Tri Dao and Albert Gu. *Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality.* arXiv:2405.21060, 2024.

[14] Wouter Kool, Herke van Hoof, and Max Welling. *Buy 4 REINFORCE Samples, Get a Baseline for Free!* ICLR workshop on Deep Reinforcement Learning Meets Structured Prediction, 2019. Author's publication record.

[15] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. *Proximal Policy Optimization Algorithms.* arXiv:1707.06347, 2017.

[16] Zhihong Shao et al. *DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models*. arXiv:2402.03300, 2024.

[17] Brendan Fong. *Decorated Cospans*. arXiv:1502.00872, 2015.

[18] Jacob Lurie. *Higher Topos Theory*. Princeton University Press, 2009. Author's text.

[19] Patrick Iglesias-Zemmour. *Diffeology*. Mathematical Surveys and Monographs, volume 185, AMS, 2013. Publisher record.

[20] Mathieu Huot, Sam Staton, and Matthijs Vákár. *Correctness of Automatic Differentiation via Diffeologies and Categorical Gluing*. arXiv:2001.02209, 2020.

[21] J. R. B. Cockett et al. *Reverse Derivative Categories*. arXiv:1910.07065, 2019.

[22] Brendan Fong, David I. Spivak, and Rémy Tuyéras. *Backprop as Functor: A Compositional Perspective on Supervised Learning*. arXiv:1711.10455, 2017.

[23] Matthijs Vákár and Tom Smeding. *CHAD: Combinatory Homomorphic Automatic Differentiation*. arXiv:2103.15776, 2021.

[24] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*, second edition. Wiley, 2006. Publisher record.

[25] Wassily Hoeffding. *Probability Inequalities for Sums of Bounded Random Variables*. Journal of the American Statistical Association 58(301), 13–30, 1963. Publisher record.

[26] Lloyd S. Shapley. *A Value for n-Person Games*. Contributions to the Theory of Games, volume II, 307–317, 1953. Publisher record.

[27] PyTorch contributors. Torch distributed launcher and published 2.7.1 installation builds. Launcher documentation. Release installation matrix.

[28] Kubernetes contributors. Schedule GPUs. Kubernetes documentation.