ELYSIA: PROVING WHAT A SPIKING NETWORK COMPUTED
A GPU-NATIVE ARCHITECTURE FOR COMPOUND SNN COMPUTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We present **ELYSIA** (**E**xplicit**ly** Parallel **S**cheduling for **I**nference **A**rchitectures), a GPU-native architecture for compound spiking computation, and its prototype **ELYSIA-P**. An optional certified backend publishes a class only once it has proved that class equals the full computation's, and abstains otherwise. That proof covers the whole temporal computation. It is the only method we tested that releases classes early while returning the full trace's answer exactly: it releases 83.75–86.31% of test classes with no class changed, and no baseline reaches it. It costs 6–15% inside the exact-dyadic conditions and $2.23\times$ per request; outside them an interval enclosure carries the same guarantee. Against our re-implementation of the strongest published early-exit method it costs 10.5–$15\times$ more per request and 173–$226\times$ batched; that method never exits at this accuracy. A truncated trace changes 150 classes; our calibrated control releases 1.46% early. On a 512-neuron model the enclosure releases all $2{,}264$ classes at median round 11 of 100, none wrong; an exact-hull rule releases all $2{,}264$ at round 12, with the same zero wrong releases, and improves coverage across four models, three passing the fp64 check. A fused LIF kernel runs a whole window in one dispatch, beating `snnTorch` and SpikingJelly by 1.14–$1.82\times$ at matched accuracy and $1.201\times$ on the whole compound task. Numerical meaning survives suspension and revision. Over 18 independent streams it retains a median 0.479 $[0.432, 0.535]$ of ELYSIA-P's throughput; no dependent stream can use it.

1 INTRODUCTION

We study when a computed result may be reused and what a changed input invalidates. We present **ELYSIA** (**E**xplicit**ly** Parallel **S**cheduling for **I**nference **A**rchitectures,[1]), a GPU-native architecture for compound spiking neural network (SNN) computation, with its own prototype **ELYSIA-P**. snnTorch and SpikingJelly are the external libraries we compare against (Eshraghian et al., 2023; Fang et al., 2023b).

A *subtask* is one stateful unit of SNN work. Its state is a membrane potential and an input cursor, and it advances that state one timestep at a time. Device state is held in *neuron groups*. A group is a fixed block of 32 neurons held together, and it is the smallest unit of state the device will place for a subtask, so a subtask occupies whole groups and never part of one. A *slot* is a group's place on the device while it is held, and the device offers L slots at once, so L is the capacity measured in groups. A *compound SNN computation* is a set of subtasks that consume each other's outputs. The result of the whole then follows jointly from all of them. Such a computation has parallel work — subtasks that do not depend on each other — which is what makes the GPU worth using. But once one set of physical resources serves several logical subtasks over time, a tensor stored on the device no longer records which subtask produced it. Nor does it record whether that subtask is still the one the caller asked for. Sharing the GPU is what breaks that bookkeeping. One subtask *consumes* another when it reads that other's output as its input, and the last subtask of a computation emits the answer, which

[1]The letters L and Y are the two in *explicitly*.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

we call the *root class*. Our spoken-digit experiment on SHD (Spiking Heidelberg Digits) shows the shape of the problem, and §3 carries it as a running example: two SNN parents, each producing a *score tensor* — one score per class, per test sample — which a *fusion head* then consumes. The fusion head is a small learned network that takes the parents' scores as its input and emits the fused scores, and a readout turns those into the root class. A suspended parent keeps its membrane history. Correcting an input restarts the descendants that consumed it (Figure 6).

Our central claim is that a spiking network can say when its own answer is already settled. A class may be published — hereafter *released* — before every spike resolves, provided a certificate proves it equals the *reference* prediction — the result the full, unabridged computation would produce. That proof costs 6–15% at exactly representable weights and 362–553× outside that regime (Table 5). The test case is a 512-neuron model whose leak leaves the dyadic conditions, the regime in which the trace is exactly representable. On its full 2,264-example test set the reset-to-zero enclosure, an interval that brackets every reachable membrane value and clears it whenever the neuron fires, releases every class at a median of 11 rounds of a 100-round budget, with zero wrong releases (Prop. 11 with Cor. 12; §5.1). Figure 2 is that claim in one panel: the comparison runs on one frozen pipeline — one trained model, one split, one fixed sample set, nothing retrained. Its other arms are a no-guarantee fast path, the published early-exit rules SEENN-I, SEENN-II and SpikeCP, a learned halting head, a calibrated margin rule and a per-sample oracle. Among the methods we tested, only the certificate returns the full trace's answer at that coverage, and it carries a proof. It executes the full trace; §5.1 separates the three senses in which it could be said to stop early. What is certified is agreement with the computation the caller asked for. The backend abstains rather than guess.

By *GPU-native* we mean that a compound request — one caller's whole computation — keeps its numerical work resident on the device. A *window* is a contiguous block of timesteps, and one window enters one *temporally fused kernel*: one device dispatch that runs every timestep of the block. The subtasks *admitted* to a block are those a capacity rule has allowed to hold device state; they are gathered into one tensor, so the device is dispatched once per window rather than once per timestep. Deciding what may run stays sequential on the CPU by design, and §5.5 reports the cost of that decision separately.

Sharing the device therefore destroys provenance in three separate ways. Waiting subtasks leave the numerical batch, so the same tensor rows serve different subtasks in different windows. The memory pool rebinds its slots as subtasks wait and resume. The fused kernel keeps the recurrence on the device, where intermediate state is not an object anyone can version. ELYSIA answers each of those three losses with an explicit object. A *dependency hypergraph* instantiates declared subtask templates and exposes the legal parallel branches. Task-private state outlives its temporary tensor slots. A committed artifact unlocks its dependents by naming the computation it came from (§3).

Two decisions are separate here, and neither constrains the other. The first is how one window of one subtask is computed numerically; we call that choice the *backend*, and we ship a step-wise LIF and a fused kernel. The second is how subtasks are placed on the device over time; we call that the *executor*, and we ship ELYSIA and ELYSIA-P. Either executor drives either backend. ELYSIA-P is this paper's own stripped prototype, built around the same fused kernel; it drops the dependency graph, checkpoints and artifact contracts, and serves streams whose subtasks have no dependencies, batch-parallel, under complete-group admission — a subtask starts only when every neuron group it needs is free. Its purpose is to separate the fused kernel's contribution from the dependency machinery's cost. With nothing for that machinery to enforce, it is the fastest configuration we measured, which sets the direction of the throughput ratios of §5.5. A dependent stream cannot use ELYSIA-P at all.

We contribute three things. First, a *certificate*: bounds computed alongside the network's own output, which prove that output already equals the full trace's. To our knowledge this is the first certificate for the temporal computation of a spiking network. Its parts are bracketing bounds on the trace — an interval per coordinate covering every reachable value — and a tight round-by-round bound. That bound localizes the undecided coordinates to a set U_K (a Jacobi iteration, the standard fixed-point sweep, from all-zero spikes). Sparse exact repair then fixes the wrong coordinates. Asynchronous window updates refine the bound. It abstains when the proof does not close, and both of its costed implementations change no class (§5.1). Second, a fused leaky integrate-and-fire (LIF) kernel that executes a whole window in one device dispatch. At exactly matched accuracy it is 1.14–1.82× faster than `snnTorch` 1.0.0 and SpikingJelly's `LIFNode` 0.0.0.0.14, each used through its pub-

lished API. It is faster still than SpikingJelly's own unmodified CuPy multi-step LIF over the same model instances (§5.4). Third, a GPU-native architecture for compound, revisable SNN computation. Its parts are dependency-linked subtasks, task-private temporal state and reusable GPU capacity under one checkpoint and artifact contract. It preserves the declared reference computation bitwise across repair and full re-execution under suspension, revision and partial publication, with replay confined to the affected subtree (Props. 1 and 2).

2 Related Work

snnTorch and SpikingJelly provide established SNN software infrastructure (Eshraghian et al., 2023; Fang et al., 2023b). We compare actual library implementations in both trained-model and matched-operator protocols. SHD supplies the public event-based learning task (Cramer et al., 2022). Parallel Spiking Neurons modify temporal dynamics to expose parallel training (Fang et al., 2023a); FPT uses an affine fixed-point representation of LIF and analyzes smooth surrogate convergence (Feng et al., 2025), which ELYSIA reuses for a different obligation. Our tight bound concerns the hard-threshold Jacobi iterate, not every parallel SNN algorithm.

Cavs separates a vertex function from an instance graph and batches neural operations along dependencies (Xu et al., 2018). Adapton tracks demanded computations to reuse results and to repair input changes consistently with fresh execution (Hammer et al., 2014). WfCommons supplies our public workflow instances (Coleman et al., 2022). ELYSIA specializes these responsibilities to SNN numerical progress. A subtask that saves its membrane and its cursor can be resumed instead of rerun, and bounds can prove a requested class before the spike trace completes. The added obligation is therefore to preserve each requested reference when subtasks are batched together, when they are suspended and resumed, and when an input is revised. Graph scheduling or incremental reuse alone does not supply that certificate, and nothing in this line owns the SNN state a revision invalidates (§3). That line also allocates computation inside one model by learning a router, whereas our admission rule is a runtime capacity rule over subtasks.

SNN inference is also stopped early by learned or statistical rules (Li et al., 2023b;c; Wang et al., 2023; Li et al., 2023a; Chen et al., 2023). Each trades accuracy for fewer timesteps and bounds reliability from calibrated data, and none proves the output is already determined. §5.1 measures every method we could configure on one frozen pipeline — one trained model, one split and one fixed sample set, with nothing retrained — where the certificate changes no class. Formal verification of early exits has been proposed precisely because an early exit can return a different answer than the full network (Elboher et al., 2025). That work verifies feed-forward early exits, where the uncertainty is over the input; ours is over the unresolved computation. Our certificate instead preserves a full-horizon model prediction while stopping its numerical solve, proving equality with the network's own reference prediction for the class it releases. Affine scans, signed linear bounds and monotone asynchronous iteration are established tools (Cousot, 1977); we specialize them to reset-dependent spike uncertainty.

Ev-Edge optimizes event representation and task mapping on commodity edge processors (Sridharan et al., 2024), while our interface addresses stateful progress and consumer contracts. GPU code-generation frameworks (GeNN (Yavuz et al., 2016)) and neuromorphic processors (Loihi (Davies et al., 2018)) target single-model execution. ELYSIA organizes compound, contract-verified computation around such kernels.

3 Method: Compound Execution

The method has two planes and two executors: a CPU control plane decides what may run, and a GPU data plane runs only what was admitted (Figure 1). We fix one request as a running example (Figure 6), the learned graph $A + B \to F \to H$, whose tensors and post-revision actions are in Appendix F.4. Three objects carry the method, and each restores one of the three kinds of provenance that sharing erases. An *execution lease*, a term from distributed systems, is the right to occupy device slots for as long as a subtask needs them. It is granted over *complete* $[32]$-neuron groups — one group holds one membrane — so a subtask never holds a partial group and its state can always be gathered back; this replaces the tensor identity that sharing destroyed. An *artifact contract* is what a producer promises the subtasks that read its output, in the sense of workflow systems; here the

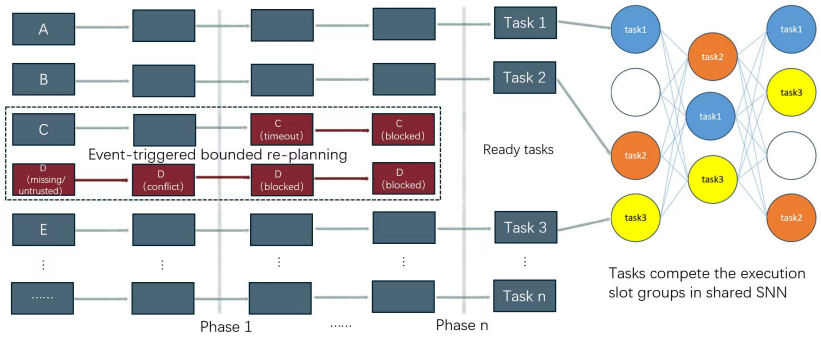


Figure 1: The shape of the compound computation, in two views of one request. *(a)* Over time, several tasks advance concurrently. A task that depends on others stays blocked until they publish. An event inside the dashed region opens a bounded re-planning step instead of restarting the computation. Such an event is a timeout, a missing or untrusted input, or a conflict. *(b)* At one instant, the ready tasks compete for the execution slot groups. Admission grants whole groups, and a task that is not admitted holds nothing. Here a group is $[32]$ neurons, and the lease forbids a partial group. The granted sizes satisfy $\sum_i m_i \leq L$ against the capacity L. Either backend executes a granted task behind one interface. Each publishes a consumer contract: the exact spike trace, or a certified class.

promise is equality with the reference readout, and it is what makes F eligible to run. This records whether a stored output is still the one the caller asked for. The *consumed lineage* of a revised input, in the sense of data provenance, is the set of artifacts whose computation read that input; this decides how far a change propagates. Revising A's input therefore invalidates $\{A, F, H\}$ and their versions, while B's committed payload is reused unchanged. Two executors sit on these objects: ELYSIA, the compound architecture, and the prototype ELYSIA-P of §1.

A *dependency hypergraph* (DHG) is a directed graph, written down before anything runs, that states how one compound computation is put together. Each node is one subtask template. A node carries AND/OR prerequisite groups, meaning which combinations of its parents are enough to run it; stage barriers; and a deterministic combination function B_v that says how the parents' outputs become its input. A selected parent set $P(v)$ and its versions are fixed before execution. Choosing whichever parents finish first instead defines a schedule-dependent computation, and that must be declared as such. Eligibility and validation are set out in Appendix F.4. A node is eligible exactly when it is pending, its stage predicates hold and every required parent version is valid. The runtime therefore never binds missing, conflicted or untrusted inputs into a child; a validator that proves only a declared property is not enough.

That interface carries two kinds of progress, and each is one transition of the runtime. An *advance* is the transition that moves a subtask's numbers forward and records what it would take to resume them. A *publish* is the transition that hands a consumer the artifact it asked for (Figure 4; Appendix F.4 carries what a safe boundary commits and how storage scales).

Admission chooses disjoint group sets for the active tasks, subject to the capacity L, so that the group sizes m_i it grants satisfy $\sum_i m_i \leq L$ (Appendix F.4). This bounds active execution slots rather than total GPU memory. Eligible tasks are then ordered by a policy, such as first-come-first-served (FCFS) or earliest-deadline-first (EDF). The order affects only the queue: a task runs when its dependencies and artifact gates allow it, not before (Appendix H).

Proposition 1 (Homogeneous lease invariance)**.** *Fix the admitted tasks and their task-local inputs, states and transitions. Suppose all temporary slots have the same execution semantics, parameters are indexed by logical neuron, and no transition reads a temporary slot identity. Replacing one injective complete allocation with another, using consistent gather/scatter, preserves every task-local transition. Thus direct group leases can replace per-slot matching without changing any committed artifact under the same input-binding and numerical conditions.*

The result authorizes elimination of mapping work; it does not apply if slot-specific parameters or learned routing change the computation, and in floating point the operand order must also be preserved (Appendix A.10).

Packing is the step that admission enables on the device. Each subtask keeps its own numbers — its membrane and its input cursor — and packing is the gathering of exactly the admitted subtasks' numbers into one active tensor batch. A waiting subtask contributes no row to that batch. Against a masked dense executor of our own, which keeps every live task in the batch, this divides the executed neuron-steps by 16 at capacity 4 and by 1.4 at capacity 64; measured throughput moves by at most 13% either way (Appendix H). The fused kernel keeps the causal LIF recurrence inside one CUDA kernel. The certified backend instead applies temporal affine scans and nonlinear gates, and exports bounds to the same artifact interface. Neither parallel execution nor zero-valued spikes makes the SNN fully linear.

Recovery follows the consumed lineage: only the subtasks that read the revised input are recomputed, so a branch that never read it is left alone (Appendix F.4).

Proposition 2 (Compositional reference preservation)**.** *Fix selected parent sets, versions and deterministic combination functions in a finite DAG. If each emitted artifact equals its declared local reference artifact and each continuation preserves private state, then every completed artifact equals the artifact the declared topology prescribes, independently of legal slot reuse and execution order.*

Take a chain $a \to b \to r$, read as: a produces, b consumes a, and r consumes b. Revising a after the chain has executed restarts b's membrane, its input cursor, its partial readout and its numerical bounds. A sibling $c \to r$ with no consumed path from a keeps its valid artifact (Appendix F.4 states this over the recorded consumed subgraph). The conditions under which every subtask eventually runs, and how a conflict's data effects interact with resource competition, are in Appendix F.4. Proofs and checkpoint requirements appear in Appendix A.6 and A.9.

4 Method: Certified Backend

This section states the optional certified backend, its guarantee and its limits. The question it answers is narrow: has the requested class already been decided, even though the spike trace is incomplete? For one neuron, fix $x_{-1} = x_{\text{init}}$, $0 \le \beta < 1$, $\theta > 0$, and an immutable input trace $I_0, \dots, I_{T-1}$. The state x_t is the *membrane potential*, called the membrane below. Define

$$u_t = \beta x_{t-1} + I_t, \qquad s_t = H(u_t - \theta), \qquad x_t = u_t - \theta s_t, \quad H(z) = \mathbf{1}[z \ge 0]. \tag{1}$$

The initial state is *post-reset*: the reset is already subtracted. Write the recurrence of Eq. (1) as a map Φ from one spike history to the next. Its reset coefficient is nonpositive, so Φ is order-reversing, and a reset-to-zero implementation needs a wider enclosure (Appendix I.6). Results apply neuronwise, after a layer's input is fixed, and do not cover recurrent candidate-dependent input. Unless stated otherwise, proofs concern real arithmetic. For binary q, the FPT affine form gives

$$A_t = \beta^{t+1} x_{\text{init}} + \sum_{j=0}^{t} \beta^{t-j} I_j, \qquad \Phi(q)_t = H\left(A_t - \theta - \theta \sum_{j<t} \beta^{t-j} q_j \right). \tag{2}$$

This is affine in a *given* spike history. The spike-generating system itself is nonlinear.

Theorem 3 (Bracketing bounds and an available certificate)**.** *Let $s^0 = 0$ and $s^{k+1} = \Phi(s^k)$. The unique fixed point s^* is the causal solution (causal: computed from earlier steps only) of Eq. (1), and*

$$s^0 \le s^2 \le \cdots \le s^* \le \cdots \le s^3 \le s^1, \qquad s_t^k = s_t^* \quad (t < \min(k, T)).$$

After exactly $K \ge 1$ evaluations, define

$$C_K = \{t : s_t^{K-1} \ne s_t^K\}, \qquad U_K = C_K \cap [K, T). \tag{3}$$

Every error in s^K lies in U_K, and $U_T = \varnothing$.

The proof sketch is in Appendix A. The raw set C_T need not be empty, and nothing here silently treats an unresolved coordinate as correct. A constant round budget cannot resolve this iteration in general. For every $0 < \beta < 1/2$ there are currents whose first exact iterate from zero is s^T, by the construction in Appendix A.2.

Theorem 4 (Certifying the artifact requested by a consumer). *Let $l \le s^* \le h$ be sound Boolean bounds and $r^* = b + Ms^*$ a fixed linear readout. With $M^+ = \max(M, 0)$ and $M^- = \min(M, 0)$,*

$$b + M^+ l + M^- h \le r^* \le b + M^+ h + M^- l. \tag{4}$$

For the candidate $\widehat{r} = b + Ms^K$ in Theorem 3, $|r^* - \widehat{r}| \le e = |M|\mathbf{1}_{U_K}$. A candidate class a is the unique reference class if $\widehat{r}_a - \widehat{r}_j > e_a + e_j$ for all $j \ne a$.

Direct pairwise margin minimization is tighter than this bound. At finite precision the readout bounds must also enclose rounding error (Appendix A.3).

Theorem 5 (Certificate-directed sparse exact repair). *Let $U = \{d_1 < \cdots < d_m\}$ contain every wrong coordinate of candidate q. Each known run $[a, b) \subseteq [0, T) \setminus U$ has the affine summary*

$$F_{a:b}(x) = \beta^{b-a} x + \sum_{j=a}^{b-1} \beta^{b-1-j}(I_j - \theta q_j). \tag{5}$$

Precompute these summaries independently. Starting at x_{init}, alternate applying each summary with a causal gate/reset at each d_i. This recovers the exact spike trace and final state, using only m sequential unresolved gates. With work-efficient segmented scans, preprocessing has $O(T)$ work and $O(\log T)$ depth, followed by $O(m + 1)$ work/depth.

The proof sketch is in Appendix A. After K certification rounds, ideal work is $O(KT + T)$ and depth is $O(K \log T + \log T + |U_K|)$. The rounding and fallback conditions that go with it are in Appendix A.

Bound updates are computed on *safe windows*: windows whose whole input is fixed, so it cannot change underneath the update. A sound enclosure $[\underline{z}(q), \overline{z}(q)]$ of every exact threshold argument supports monotone bound updates from $(l, h) = (0, 1)$:

$$l' = \max\{l, H(\underline{z}(h))\}, \qquad h' = \min\{h, H(\overline{z}(l))\}.$$

Scanning the candidate-dependent affine summary of a window gives boundaries consistent with the *same* global candidate; stale independent serial window solves are not an exact fallback. Agreement of rounded iterates is insufficient in finite precision, and nonzero-width enclosures can stall at ties, so exact T-round termination does not transfer automatically (Appendix F.2).

Theorem 6 (Asynchronous selected-window bounds). *Initialize the* published bounds*, the ones any reader of this task may rely on, to $(l, h) = (0, 1)$. An update may select any coordinates and read any previously published sound lower/upper bounds for the same fixed task, even from different event versions. It proposes $H(\underline{z}(\widetilde{h}))$ and $H(\overline{z}(\widetilde{l}))$ at those coordinates and atomically merges them into the* current *bounds by max/min. The published box (the enclosure in every coordinate) remains sound and shrinks monotonically regardless of delay. With exact evaluations, all coordinates eventually resolve if each unresolved coordinate is updated repeatedly and each predecessor's read-version index tends to infinity along those updates.*

This is a specialization of monotone asynchronous fixed-point reasoning (Cousot, 1977): it permits suspension between selected-window updates without a task-wide Jacobi barrier. It does not inherit the synchronous K-prefix certificate. The proof sketch is in Appendix F.2.

5 Evaluation

We evaluate four questions. Does execution preserve the reference? Learned compound graphs, injected faults and revision studies (§5.2–5.3). What accelerates execution? Matched component and serving runs (§5.4, §5.5). When does selective recovery help? Replay against whole-root re-execution, with a width sweep (Appendix H). When does certification help? A calibrated stopping rule on the same information budget, costed per request (§5.1). Comparisons keep their own models and timing boundaries; ratios are never multiplied across them.

5.1 Numerical release policy

We first ask what early release costs in answers. Figure 2 puts every method we could configure on one *frozen pipeline* — one trained model, one dataset split and one fixed set of test samples, with no

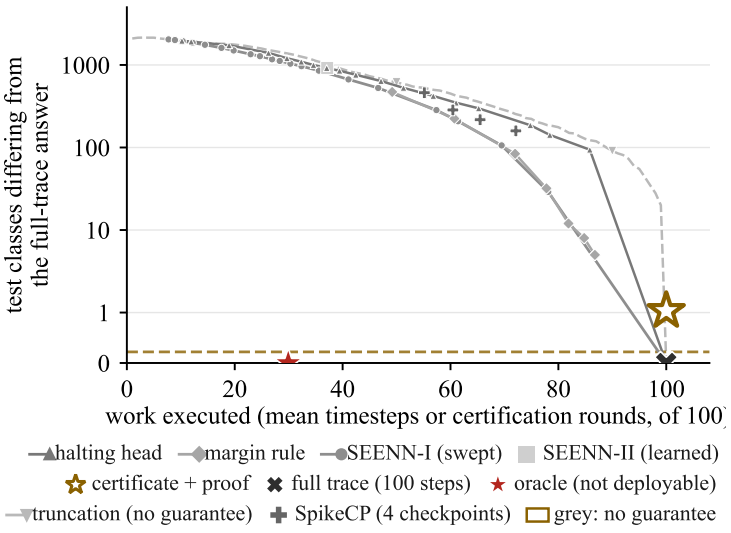


Figure 2: On one frozen pipeline, every early-release method we swept trades the answer for work except the certificate and our own margin control. The certified path still executes the full trace, so its marker sits with the full trace. What it carries there is a proof of equality. The vertical axis counts test classes that differ from the full trace's answer, so zero means the answer is preserved exactly. The plotted certificate marker is the *outward-rounded FP32/FP64 enclosure* arm — an interval arithmetic run in which every operation is rounded away from the true value, so the interval can only over-cover — which is our most expensive route to the guarantee. It costs $362\times$ one bare fused full-trace request as a complete per-request service call, and $553\times$ the whole test split against our own fastest no-guarantee arm, which sits at a *different* accuracy (0.751325 against the certificate's 0.769435). The cheaper *exact-dyadic class scan* — the same scan run where the arithmetic happens to be exact — is not plotted, because it belongs on a different horizontal axis; it is measured on its own RTX 4060 Laptop boundary in Table 32 and at the vote-inclusive boundary of Table 5. It changes no class either, so the plotted point is an upper bound on the guarantee's cost. Appendix F.1 separates the three senses of the certification round and reports the cost of each.

retraining anywhere in the comparison — at the best setting of each method's own *knob*, meaning the one hyperparameter that method exposes (Tables 17, 13). Each method we swept gains speed at the cost of changed classes, and no setting of any of them reaches the certificate's coverage with zero changed classes. Our own calibrated margin-rule control does reach zero, on 1.46% of the split against 83.75–86.31% for the certificate, and the per-sample oracle exits at a mean of 29.9 timesteps of 100 while changing nothing.

The certificate does both, and the guarantee costs little inside the exact-dyadic conditions — the regime where every weight and the leak are exactly representable as $k/2^n$, so the trace itself is exact. Our certified backend has three costed variants on the same full-trace reference, and Table 5 is their whole cost. *Exact-trace certification* runs the trace and proves the released class equals the full computation's. That proof costs 6% per request and 15% batched, which is the 6–15% of our claim. The *exact-dyadic scan* adds early release at $2.23\times$ per request and $7.8\times$ batched, releasing every class at a median certification round of 3 of 64. The *outward-rounded FP32/FP64 enclosure* keeps the same guarantee where no exact dyadic representation exists, at $362\times$ a single request and $553\times$ the whole split. The scan is cheaper: $6.8\times$ the certified full trace batched.

The exact-dyadic route needs a leak that is a power of two, one hard-coded current unit, $\theta = 1$ and an exact-integer readout on a fixed T (Appendix G.8 lists all six conditions, and why the $\beta = 19/20$ SHD model falls outside them). Those two multipliers come from leaving the conditions, not from the guarantee itself.

Within those conditions, release is decided by the rule a model runs, and a better rule releases more. Our new one-step rule, the *hull rule*, returns the exact interval hull of the reachable membrane range over the current box, where the rule this paper ships widens the interval instead. No sound rule

	Sequential-sweep rule	Hull rule (exact one-step)
Rounds run	60	36
Classes released	499	**2,264**
Never released	1,765	**0**
Median certification round	54.0	**12.0**
Mean / p90 certification round	52.33 / 59.0	13.51 / 20.0
Round-budget seconds, rounds × per round	403.8	318.2
Seconds per round	**6.73**	8.84
Wrong pins / wrong certified classes	0 / 0	0 / 0
Enclosure lower / upper violations	0 / 0	0 / 0

Table 1: Our two release rules on the same deployed 512-neuron recurrence ($\beta = 19/20$, $\theta = 1$, $T = 100$, reset-to-zero, the official 2,264-sample test split), measured under one exclusive GPU block with 60.0 s of confirmed peer-free settling; gates first, the deployed FP32 accuracy equals the float64 reference at $1{,}742/2{,}264 = 0.769434628975265$ with 0 class and 0 spike disagreements over 115,916,800 coordinates. A coordinate is *pinned* once the certificate's two bounds agree on it, so its bit is settled. A released-class count on this recurrence names its rule: the unknown-tail channel 0, the shipped sequential-sweep rule 499 in 60 rounds, and the enclosure path of Table 34 all 2,264 at median round 11. The hull rule's dominance here is aggregate, not coordinate-wise: 93 of 2,782,003,200 coordinate-rounds over 24 rounds go the other way (Appendix I.2). The comparator is this paper's own sequential-sweep rule, so the table is an internal comparison: the hull rule pays 1.31× more per round and returns it in rounds, releasing all 2,264 where our sequential-sweep rule leaves 1,765. The round-budget row is the product of the rows above it, not an independent measurement.

whose whole state is that box resolves more, and this paper's own rules carry an enclosure beside it (Appendix I.2).

It is a component improvement to our own release rule, and it is costed as one: it spends 1.31× more per round, 8.84 s against 6.73 s. It runs 36 rounds instead of 60, releasing all 2,264 classes at a median round of 12, against 499 at a median round of 54 under our sequential-sweep rule, which leaves 1,765 never released. The gain is coverage: 36 rounds at 8.84 s is less total round budget than 60 at 6.73 s, and Table 1 gives both. At the shared budget $K = 30$ it releases at least as many roots as our sequential-sweep rule does, across four synthetic models spanning leak, width and horizon; the fp64 soundness gate passes on three (Appendix I.2).

The certificate's certification round is when its proof closes, and Appendix F.1 separates its three senses on the same 2,264 roots. The round-by-round release decision costs more depth than the serial trace at the paper's own scales (Appendix F.1 states it with its constants).

The guarantee is forced rather than earned — the margin test never reads a label (Prop. 8, at accuracy 0.00 by construction) — and it survives depth and precision. On a two-layer 512–512 chain it releases $2{,}201/2{,}264 = 97.22\%$ at 0 wrong against both the exact fp64 reference and the deployed fp32 argmax, and a three-layer chain matches it. On 4,157 N-MNIST samples the same conditions release 98.82% in fp64 and 97.93% in fp32, with 0 wrong releases either way (Appendix F.1). Releasing each root as it is proved rather than running the certificate to its round budget saves 1.62× at identical released classes and rounds on all 2,264 roots, so the costs above belong to that other configuration. On a published-configuration SHD trunk the certificate releases 2,007 of 2,264 at 0.92182, none wrong (Appendix I.3).

5.2 Execution preserves the reference

A spoken-digit computation with two non-equivalent SNN parents and a trained fusion head reaches 1,301/2,264 correct test predictions (57.46%). It runs with the fused kernel and with a CuPy multi-step LIF from SpikingJelly. Every test root completes, all 568 repeated failures recover, and the 426 pairs that compare output quality and the 120 pairs that compare timing are bitwise identical in artifacts, spikes, membranes and predictions (Table 6).

5.3 Local recovery and evidence lineage

A second learned graph extends the same shape by two further consumption levels, so its root depends on a four-subtask chain rather than a two-subtask one; that model is frozen at a test accuracy of $1{,}530/2{,}264$ (67.58%). With it, 568 full-test and 400 timed-arm comparisons of local lineage repair against affected-root restart match fresh scores, spikes and membranes exactly. The wall-clock saving, however, is not established there (Appendix E.1).

5.4 Kernel and component timing

The comparison fixes inputs, weights, state, graph and recovery, so the two arms differ only in the numerical path. Against SpikingJelly's CuPy multi-step LIF the fused LIF is a median $1.521\times$ and $1.874\times$ faster, with all 270 measured model instances matching in spikes, membranes and outputs. The complete-task wall ratio is $1.201\times$ (Table 6). Against independently authored runtimes at identical accuracy it is 1.1–$1.8\times$; a graph of our *own* step-wise loop wins on these instances (Appendices B, I.5 and I.9).

5.5 Serving throughput

One serving loop drives every arm on identical SHD streams, at capacities 4 and 16 under FCFS or EDF. Over 18 independently generated streams ELYSIA retains 0.479 $[0.432, 0.535]$ of ELYSIA-P's throughput at capacity 4 and 0.650 $[0.587, 0.694]$ at capacity 16 — the measured cost of carrying a dependency graph, task-private checkpoints and artifact contracts on a stream with nothing for them to act on. Where each executor keeps its own kernel instead, as shipped, the contrast reverses: 1.249 $[1.225, 1.298]$ at capacity 4 and 1.124 $[1.033, 1.196]$ at capacity 16 on the same 18 streams, and on the RTX 5090 ELYSIA wins all six device-and-capacity combinations and beats ELYSIA-P on all 30 paired repeats (median 1.172, minimum 1.027, sign test $p = 2^{-30}$).

On a $4{,}800$-node dependent graph a revision rebinds 4–8 nodes against 75–150 under recompute-from-scratch, and the system is 1.21–$1.24\times$ faster per revision at every density tested. Against a hand-written memo of already-computed results, walking the dependent chain to find what to reuse, it is 1.06–$1.08\times$ slower: keeping the consumed lineage and the artifact contract current costs 0.246–0.250 ms per revision against 0.010–0.018 ms. End to end the advantage survives median root completion at $d = 1$ (0.965) and ties at $1/2$, but loses at $1/8$ (0.838) and $1/32$ (0.753); the crossover is measured. No arm above carries together what the contract provides — exact-lineage invalidation, continuation instead of restart, and a published class that stays the reference computation's — and Appendix H.1 shows that its cost comes from the work, not from how the arm is written. Appendix F.5 carries the fused kernel's 2.25–$3.61\times$ end-to-end lead and the detail.

6 Discussion and Conclusion

Our measurements confirm the claim. Repair and full re-execution agree bitwise, and replay tracks the affected subtree, up to $64\times$ less work at 64 branches. The certified backend publishes a class only against a proof of equality with the full computation, and was the only such arm. Gate on declared artifacts, keep state private per task, and invalidate consumed lineage on revision.[2]

[2]Both learned models are frozen and use SHD; full scope is Appendix G.2.

REPRODUCIBILITY STATEMENT

Complete proofs appear in Appendix A. The source package contains the reference checks, experiment scripts, recorded results, an evidence ledger, and commands. Each result is identified as a new execution or a historical record. The package includes frozen historical runners and restored CSV/JSON evidence, new exact numerical implementations, the integrated direct-lease path, and a complete learned multi-parent execution package. Reported later measurements are distinguished from locally recomputed archived statistics. The locally recomputed learned-graph numbers are separate studies and are never substituted into any other table. Dataset terms: SHD is used under its Creative Commons Attribution 4.0 International release (Cramer et al., 2022) and N-MNIST from its public release (Orchard et al., 2015).

AI USE STATEMENT

AI tools assisted manuscript revision, mathematical derivation and checking, code generation, and experiment auditing. The package records the checks actually performed. These checks do not replace author verification of proofs, originality, measurements, or submission requirements.

ETHICS STATEMENT

This work develops a GPU-native execution architecture for spiking neural networks and uses only public datasets (SHD and N-MNIST) and synthetic traces; no human subjects, personal data or animals are involved, and no model is deployed in a safety-critical setting. The certified backend can withhold a class when its guarantee does not hold, so the system fails closed rather than publishing an unproven class.

REFERENCES

Jiechen Chen, Sangwoo Park, and Osvaldo Simeone. Knowing when to stop: Delay-adaptive spiking neural network classifiers with reliability guarantees. *arXiv preprint arXiv:2305.11322*, 2023. URL https://arxiv.org/abs/2305.11322.

Tainã Coleman et al. Wfcommons: A framework for enabling scientific workflow research and development. In *Workshop on Workflows in Support of Large-Scale Science*, 2022.

Patrick Cousot. Asynchronous iterative methods for solving a fixed point system of monotone equations in a complete lattice. Technical Report IMAG-RR-88, Université Scientifique et Médicale de Grenoble, 1977. URL https://cs.nyu.edu/~pmc309/COUSOTpapers/IMAG-RR88.shtml.

Benjamin Cramer et al. The heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

Mike Davies et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

Yizhak Yisrael Elboher, Avraham Raviv, Amihay Elboher, Zhouxing Shi, Omri Azencot, Hillel Kugler, and Guy Katz. Bridging efficiency and safety: Formal verification of neural networks with early exits. *arXiv preprint arXiv:2512.20755*, 2025.

Jason K. Eshraghian et al. Training spiking neural networks using lessons from deep learning. *Proceedings of the IEEE*, 111(9):1016–1054, 2023.

Wei Fang, Zhaofei Yu, Zhaokun Zhou, Ding Chen, Yanqi Chen, Zhengyu Ma, Timothée Masquelier, Tiejun Huang, and Yonghong Tian. Parallel spiking neurons with high efficiency and ability to learn long-term dependencies. In *Advances in Neural Information Processing Systems*, volume 36, 2023a.

Wei Fang et al. Spikingjelly: An open-source machine learning infrastructure platform for spike-based intelligence. *Science Advances*, 9(40), 2023b.

Wanjin Feng, Xingyu Gao, Wenqian Du, Hailong Shi, Peilin Zhao, Pengcheng Wu, and Chunyan Miao. Efficient parallel training methods for spiking neural networks with constant time complexity. *arXiv preprint arXiv:2506.12087*, 2025.

Matthew A. Hammer, Yit Phang Khoo, Michael Hicks, and Jeffrey S. Foster. Adapton: Composable, demand-driven incremental computation. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 156–166, 2014. doi: 10.1145/2594291.2594324. URL https://matthewhammer.org/adapton/.

Yuhang Li, Tamar Geller, Youngeun Kim, and Priyadarshini Panda. Seenn: Towards temporal spiking early-exit neural networks. *arXiv preprint arXiv:2304.01230*, 2023a. URL https://arxiv.org/abs/2304.01230.

Yuhang Li et al. Dynamic confidence-aware spiking neural network. In *IEEE/CVF International Conference on Computer Vision*, 2023b.

Yuhang Li et al. Input-aware dynamic timestep spiking neural networks for efficient in-memory computing. In *Design Automation Conference*, 2023c.

Garrick Orchard, Ajinkya Jayawant, Gregory K. Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience*, 9:437, 2015. URL https://doi.org/10.3389/fnins.2015.00437.

Shrihari Sridharan, Surya Selvam, Kaushik Roy, and Anand Raghunathan. Ev-edge: Efficient execution of event-based vision algorithms on commodity edge platforms, 2024. URL https://arxiv.org/abs/2403.15717.

Théotime Wang et al. Learning delays in spiking neural networks using dilated convolutions with learnable spacings. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2306.17670.

Yuhang Wang et al. Anytime inference with cutoff in spiking neural networks. 2023.

Shizhen Xu, Hao Zhang, Graham Neubig, Wei Dai, Jin Kyu Kim, Zhijie Deng, Qirong Ho, Guangwen Yang, and Eric P. Xing. Cavs: An efficient runtime system for dynamic neural networks. In *USENIX Annual Technical Conference*, pp. 937–950, 2018. URL https://www.usenix.org/conference/atc18/presentation/xu-shizen.

Esin Yavuz, James Turner, and Thomas Nowotny. Genn: a code generation framework for accelerated brain simulations. *Scientific Reports*, 6:18854, 2016.

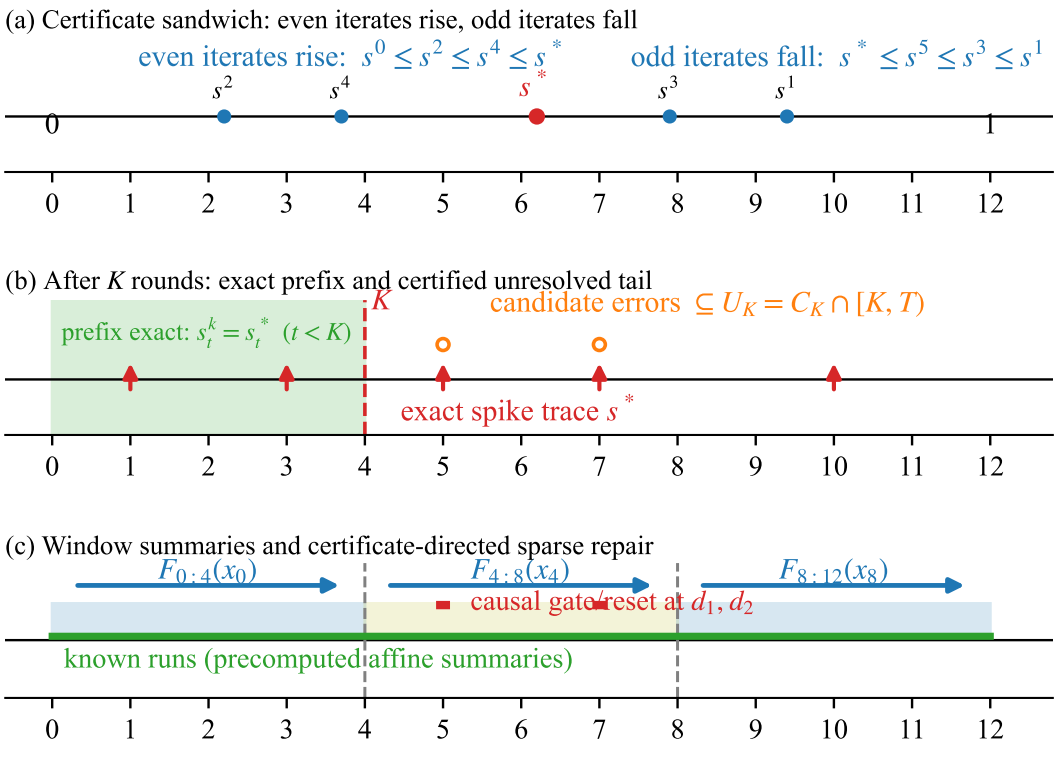


Figure 3: The optional temporal backend in one picture. (a) Certificate sandwich (Theorem 3): even iterates rise and odd iterates fall around the causal solution s^*. (b) After K rounds, every coordinate $t < K$ is exact; remaining candidate errors lie in $U_K = C_K \cap [K, T)$. (c) Window summaries transport post-reset boundary states; certificate-directed repair (Theorem 5) re-gates only the unresolved coordinates d_i, replaying precomputed known runs in between.

A Complete Proofs and Reference Procedures

All real-arithmetic statements use Eq. (1), a fixed finite trace, $0 \le \beta < 1$, $\theta > 0$, and initial post-reset state x_{init}. No continuity, differentiability, stochastic assumption, or contractivity of the hard threshold is assumed.

Theorem 7 (Tight worst case for this Jacobi iteration)**.** *For any $T \ge 1$ and $0 < \beta < 1/2$, choose $\beta^2/(1-\beta) < c < \beta$, $x_{\text{init}} = 0$, $I_0 = \theta(1+c)$ and $I_t = (1-\beta)\theta(1+c)$ for $t > 0$. Then $\Phi(q)_0 = 1$ and $\Phi(q)_t = 1 - q_{t-1}$ for $t > 0$. The first exact iterate from zero is s^T.*

Proof sketch. Here $A_t = \theta(1+c)$. If $q_{t-1} = 0$, older resets sum to less than $\beta^2/(1-\beta) < c$; if $q_{t-1} = 1$, the reset sum is at least $\beta > c$. Thus the fixed point alternates, while each round corrects exactly the next prefix position. Appendix A.2 gives the full induction. □

Small leak alone therefore cannot justify constant K. This is a lower bound for the stated iteration, not for every parallel LIF algorithm.

A.1 Affine form, uniqueness, sandwich and prefix

For a candidate q, define

$$x_t[q] = \beta^{t+1} x_{\text{init}} + \sum_{j=0}^{t} \beta^{t-j}(I_j - \theta q_j), \qquad x_{-1}[q] = x_{\text{init}}.$$

Induction gives $x_t[q] = \beta x_{t-1}[q] + I_t - \theta q_t$. The threshold argument $z_t(q) = \beta x_{t-1}[q] + I_t - \theta$ is exactly the expression in Eq. (2). Every coefficient of q is nonpositive, so $q \le q'$ implies $z(q) \ge z(q')$ and $\Phi(q) \ge \Phi(q')$.

The causal solution is a fixed point by substitution. Conversely, its zeroth coordinate is fixed by I_0, x_{init}; if the coordinates below t are fixed, the strictly lower triangular formula fixes coordinate t. Induction proves uniqueness. Since Φ^2 is monotone and zero is the least Boolean vector, $s^0 \le s^2$ and repeated application gives increasing even iterates. Applying Φ gives decreasing odd iterates. Starting with $s^0 \le s^* \le \Phi(0) = s^1$ and applying order reversal repeatedly places s^* between each consecutive pair. This does not rely on a claim that an arbitrary antitone map has no two-cycle: the causal structure supplies uniqueness and finite convergence here.

To prove prefix exactness, assume $s^k_t = s^*_t$ for $t < k$. Every coordinate $t \le k$ of $\Phi(s^k)$ reads only $j < t \le k$, hence agrees with the reference. The base $k = 0$ is vacuous and establishes coordinate zero at round one. Thus $s^T = s^*$, including threshold ties with $H(0) = 1$. If two adjacent bounds agree at t, the reference equals that bit. Every other potential error is in C_K, and prefix exactness removes $t < K$. This proves Theorem 3. Refined endpoints can be stored as $l_K = \min(s^{K-1}, s^K)$ and $h_K = \max(s^{K-1}, s^K)$, replacing both by s^K_t for $t < K$. They obey $h_K - l_K = \mathbf{1}_{U_K}$.

Sandwich and prefix: the main text's proof sketch. Nonpositive reset coefficients make Φ order-reversing. Strict lower triangularity fixes each coordinate from earlier ones, giving uniqueness and the exact-prefix induction. Applying Φ^2 from zero gives increasing even iterates and decreasing odd iterates around s^*. Agreement of adjacent bounds certifies a coordinate; the prefix certifies every $t < K$ independently.

A.2 Tight low-leak construction

The interval for c is nonempty because $\beta^2/(1-\beta) < \beta$ when $0 < \beta < 1/2$. The inputs in Theorem 7 give $A_t = \theta(1+c)$ for every t. At $t > 0$, if $q_{t-1} = 0$ the reset history is at most $\sum_{d=2}^{t} \beta^d < \beta^2/(1-\beta) < c$, so $z_t(q) > 0$. If $q_{t-1} = 1$ the history is at least $\beta > c$, so $z_t(q) < 0$. Also $z_0 = \theta c > 0$. Consequently $\Phi(q)_0 = 1$ and $\Phi(q)_t = 1 - q_{t-1}$. The fixed point is $(1, 0, 1, 0, \ldots)$. Inductively, after round k, $s^k_t = s^*_t$ for $t < k$ and $s^k_t = k \bmod 2$ for $t \ge k$. If $k < T$, coordinate k differs from $s^*_k = (k+1) \bmod 2$. Thus the first exact iterate is T. For this construction $C_T = \{T-1\}$ while $U_T = \varnothing$. The rational choice $\beta = 1/4, c = 1/8, \theta = 1$ is used in the checks.

Readout bounds: the sentence the main text carries. The proof minimizes each signed linear term over $[l, h]$.

A.3 Readout and class-valued artifacts

If $l_t \le s^*_t \le h_t$, a positive coefficient multiplies its minimum at l_t and a negative coefficient at h_t. Summing these endpoints proves Eq. (4). Since s^K agrees outside U_K, $|s^* - s^K| \le \mathbf{1}_{U_K}$; applying the triangle inequality gives $|r^* - \widehat{r}| \le |M|\mathbf{1}_{U_K}$. For class a and competitor j, $r^*_a - r^*_j \ge \widehat{r}_a - \widehat{r}_j - e_a - e_j$. Strict positivity establishes the unique reference class. A sharper test directly minimizes the pairwise margin over the same box. For $d = M_a - M_j$, it is sufficient that

$$b_a - b_j + \sum_{t:d_t \ge 0} d_t l_t + \sum_{t:d_t < 0} d_t h_t > 0 \quad \text{for every } j \ne a.$$

The bounds are sufficient, not necessary: they ignore dependence among the unresolved spikes. Fixed sum/mean readouts are included by vectorizing time and neuron coordinates. For floating-point readouts, enclose the summation and product error as well; ordinary rounded logits are not exact bounds.

A.4 Shared numerical producers

Suppose an immutable graph manifest supplies a verified identity $s^*_e = Q_e s^*_p$ for every consumer edge e of producer p. Here Q_e copies or selects coordinates and may insert fixed zero coordinates, so it is order-preserving and maps Boolean vectors to Boolean vectors. If $l_p \le s^*_p \le h_p$, then $Q_e l_p \le Q_e s^*_p \le Q_e h_p$ immediately. Theorem 4 applied to these endpoints therefore certifies the consumer's own readout. Equivalently its readout on the producer is $b_e + M_e Q_e s^*_p$. Neither equality of the readouts nor their invariance under Q_e is required. The immutable committed class remains correct while other consumers request further producer progress. Retiring the shared state

is legal after every declared consumer has committed; adding a future consumer requires retaining or recomputing the state. Sharing different inputs without a proved transform identity is outside this rule.

For the SHD delay example, let $(D_d v)_t = 0$ for $t < d$ and v_{t-d} otherwise, truncated to horizon T. The LIF encoder has zero initial state, positive threshold and time-invariant parameters. Its zero-input prefix has zero membrane and no spikes. Induction after that prefix gives $s^*(D_d I) = D_d s^*(I)$, including exact threshold ties. For candidates with the same zero prefix, the affine recurrence similarly gives $\Phi_{D_d I}(D_d q) = D_d \Phi_I(q)$: the first d candidate states vanish and each subsequent state equals the corresponding unshifted candidate state. Thus every Jacobi round can be shared. If the base exact-prefix length is k, the transformed enclosure additionally fixes the first $\min(T, k + d)$ coordinates. The three time-block readouts must still be evaluated separately. This identity is particular to the declared inputs, initialization and parameters; it does not eliminate arbitrary compound dependencies.

A.5 Sparse repair and suffix fallback

Let the unresolved indices be sorted as $d_1 < \cdots < d_m$. The complement splits into $m + 1$ known runs, allowing empty beginning or ending runs. Within each such run, $q_j = s^*_j$ by the certificate. Unrolling the recurrence gives Eq. (5) for any entering state. Starting from the true initial boundary, the first summary therefore returns the true next boundary. At the following unresolved position, a causal gate/reset sees its true pre-reset membrane and returns its reference bit/state. Induction over the alternating runs and positions proves equality of all spikes and final state.

For complexity, represent step j of a known run by affine pair $(\beta, I_j - \theta q_j)$. Applying (b, h) after (a, g) composes to $(ba, bg+h)$, an associative operation. Work-efficient segmented scans summarize all runs in $O(T)$ work and $O(\log T)$ depth. Applying the $m + 1$ summaries and m unresolved gates costs $O(m + 1)$ additional work/depth. The empty-U case still reconstructs a final state if it is requested. The implementation in the rational checker computes summaries with Python loops; it verifies the algorithm, not the claimed ideal parallel depth.

A simpler exact suffix fallback starts at $p = \min U$ (or T if empty). The prefix is exact, and its true post-reset boundary is

$$x_{p-1} = \beta^p x_{\text{init}} + \sum_{j<p} \beta^{p-1-j}(I_j - \theta q_j).$$

Running the original recurrence from p recovers the reference by causal induction. An uncertified window boundary cannot substitute for this boundary. Counterexample: $\beta = 1/2$, $\theta = 1$, $I = (1, 0.6)$, $x_{\text{init}} = 0$, and singleton windows. True spikes are $(1, 0)$, whereas independently solving both windows with the zero-iterate boundaries $(0, 1)$ gives $(1, 1)$. For a serial floating-point reference, a reordered boundary reconstruction may change a threshold decision; full reference replay is safe without an additional numerical equivalence proof.

Sparse repair: the main text's proof sketch. A known run uses reference spikes and hence maps the true incoming boundary to the true outgoing boundary. The next unresolved gate then sees its true input. Induction over runs and unresolved gates establishes exactness. A parallel summary need not match serial floating-point rounding, and full causal replay under the requested backend conditions is the conservative implementation fallback when validated arithmetic is unavailable.

A.6 Window equivalence and complete checkpoints

For $t = a_w + r$ and candidate q, expand the local recurrence from $X_w(q)$:

$$z_{a_w+r}(q) = \beta^{r+1} X_w(q) + \sum_{j=0}^{r} \beta^{r-j} I_{a_w+j} - \theta - \theta \sum_{j=0}^{r-1} \beta^{r-j} q_{a_w+j}.$$

Substitution of the scanned boundary gives the global $z_t(q)$. This proves windowed/global equality in real arithmetic for arbitrary nonempty contiguous partitions, including unequal lengths. The offset $g_w(q)$ excludes the incoming-boundary contribution; calling it the full post-reset state would double-count that contribution. A reference round is:

Execution procedure: numerical progress and artifact progress.
Input: graph, fixed parent bindings, artifact contracts, active-group capacity.

1. Admit eligible nodes; freeze each node's input/parameter/parent-version identity.
2. Lease complete groups; gather the corresponding private numerical checkpoints.
3. Advance to a safe boundary; accept the complete checkpoint only under current bindings and a valid lease generation.
4. Publish each proved contract under valid versions; enable a consumer when all its gates hold and retain producer state for outstanding requests.
5. Serialize source revisions with commits and consumption; invalidate affected consumed descendants, reject old updates and discard their checkpoints and certificate queues. Rebind and restart after new parents become valid; retain independent artifacts.
6. Return each current-version root when its contract holds; retire a producer only when all its live requests are satisfied.

Figure 4: One architecture with interchangeable numerical components. Steps 2–3 preserve spiking history; step 4 connects numerical guarantees to dependency progress. Step 5 discards progress tied to obsolete inputs. Checkpoints and artifact contracts determine which progress can be retained through lease reuse and input revision.

1. Initialize $s^0 = 0$ and scan $g(s^0)$ to obtain $X^0 = X(s^0)$.
2. From stored (s^k, X^k), evaluate every local threshold argument and write s^{k+1}. Do not read any partially updated neighboring slice.
3. Form C_{k+1} from the two iterates and, in exact arithmetic, refine it to $U_{k+1} = C_{k+1} \cap [k+1, T)$.
4. Compute $g(s^{k+1})$ and scan it to obtain X^{k+1}.
5. Commit the new state atomically at the round boundary; return bounds, emit a contract-certified artifact, repair, or yield.

For validated bounds, the lower and upper iterates and their interval boundaries replace the exact-iterate state. Ordinary rounded code must not apply the exact-prefix refinement without verifying its decisions.

Let a complete checkpoint contain all mutable state needed by a deterministic transition $F_{\mathcal{B}}$, plus identities of immutable inputs and parameters. The conditions $\mathcal{B}$ record precision, backend, arithmetic/operand order, relevant batch layout and device settings, and RNG state if randomness is used. Restoring the same checkpoint gives the same next state by determinism; induction gives the same bit patterns at every future safe boundary. This is the deterministic replay proposition. It is not a theorem about another backend, another rounding order, or a partially committed GPU round. Window boundaries can be omitted if deterministically recomputed at an explicitly counted cost.

A.7 Validated rounding and nontermination at ties

Suppose $[\underline{z}(q), \overline{z}(q)]$ encloses the exact real threshold arguments for every Boolean q, including errors in inputs, powers, products, sums and boundary terms. Let $l \le s^* \le h$ be decision sequences ordered componentwise. Two facts then chain: each enclosure brackets its own exact argument, and z is antitone in the decision sequence, so the order reverses. Reading the chain left to right,

$$\underbrace{\underline{z}(h)}_{\text{lower bound}} \le \underbrace{z(h) \le z(s^*) \le z(l)}_{\text{antitonicity}, l \le s^* \le h} \le \underbrace{\overline{z}(l)}_{\text{upper bound}} .$$

Applying monotone H shows that $H(\underline{z}(h))$ is a sound lower bound and $H(\overline{z}(l))$ a sound upper bound. Max/min with the old sound bounds preserves soundness and monotonically shrinks the box. Initialization $(0, 1)$ is sound, so induction proves the invariant for all simultaneous updates. Only coordinates where bounds agree are certified. An upper endpoint of zero must yield one because $H(0) = 1$.

This theorem is conditional on a sound evaluator. Rational intervals in the check script satisfy that condition by exact arithmetic; they are not a GPU outward-rounding implementation. If the exact

argument is zero and an evaluator keeps returning $[-\epsilon, \epsilon]$, bounds remain $(0, 1)$ forever. Stagnation does not certify unresolved positions, and finite-T termination requires exact decisions or a separately verified tie-handling reference.

A.8 Asynchronous safety and eventual exact resolution

Index atomic merge events by n. An update selects a coordinate set and reads previously published predecessor bounds $\widetilde{l}, \widetilde{h}$. They may be stale or coordinatewise assembled, but every component must bound the same immutable reference s^*. Thus $\widetilde{l} \le s^* \le \widetilde{h}$. The enclosure proof gives sound lower and upper proposals. Merging with current bounds preserves $l^n \le l^{n+1} \le s^* \le h^{n+1} \le h^n$ even if proposals arrive out of order. A plain overwrite of an older box is not the stated monotone merge.

For convergence with exact evaluation, require that each unresolved coordinate is updated infinitely often in a nonterminating execution. For each predecessor $j < t$ and fixed event index N, all sufficiently late updates of t must read both predecessor bounds from versions at least N. This is the explicit freshness condition; bounded delay suffices. Coordinate zero reads no spikes and resolves at its first update. Assume all predecessors below t stabilize after finite event N. Freshness and repeated updates guarantee a later update of t reads all of them at their reference values. Its upper and lower exact threshold arguments then coincide, resolving t. Finite induction over T gives a finite completion event, without a bound on repeated wasted updates. If approximate enclosures are used, they must additionally become decisive, or a sound exact fallback must be available. Arbitrarily small positive-width intervals cannot resolve every exact tie.

The reference summary tree computes affine compositions in causal order using two accumulators for prefix queries. It updates only changed leaves and their ancestors. An order-statistic tree stores one active bit per window, selecting a cyclic successor by prefix count or a random active rank; each selection and removal costs $O(\log W)$. Only a merged length-L window is rechecked for resolution. Complete snapshots include all trees, the FIFO proposal queue, bounds, version counter, cursor and RNG state. Global reference assertions and full serialization are audit operations, excluded from the local kernel complexity. The reference evaluates both local proposals from the same captured local slices and tree versions. The evaluated prefix and local slices must constitute sound candidates; a concurrent implementation needs version consistency during tree queries.

A.9 Composition over the recorded consumed subgraph

Fix a finite DAG. Each node has an immutable selected parent set $P(v)$, deterministic input combination function B_v, artifact function A_v, and external input. Selected parent artifacts must be validated and version-stable before binding. The local backend either computes the common declared reference exactly or proves that the emitted artifact equals A_v of that reference; its state remains private and its continuation preserves the local contract.

Induct in topological order. Source nodes share their external input with the reference. For an internal node, all selected parents equal the reference by induction, so B_v produces the same bound input. Its local exact solver or sound artifact certificate produces the reference artifact. This establishes equality of every completed artifact for any legal schedule and capacity. The guarantee is conditional on the same $P(v)$: a schedule-dependent choice can produce different legitimate computations. Class-only certification applies only if A_v exposes the class; a child reading full logits needs a stronger artifact contract. Nonlinear semantic outputs are not covered automatically.

For liveness, assume validation eventually succeeds for required inputs, each local solve terminates, capacity can admit a complete required group, and each continuously ready task is eventually served sufficiently often. A source then finishes; by induction each node whose selected parents finish eventually becomes ready and finishes. Fairness without feasible admission or eventual validation would not suffice. The graph edge relation confines data lineage effects of a conflict to descendants, but it does not confine timing effects when unrelated nodes share resources.

A.10 Complete homogeneous leases and generation safety

Let $J = \{(i, j) : i \text{ admitted}, 1 \le j \le m_i\}$ denote logical entries, and let $\pi : J \to \{1, \dots, L\}$ be injective. Gathering by π maps logical data, parameters and state to their execution positions;

scattering uses the inverse map on the image of π. For any other injection ρ, there is a permutation P of the temporary slots mapping $\pi(J)$ to $\rho(J)$ and their corresponding entries. Under the stated homogeneous contract, the packed transition satisfies $F(Pz) = PF(z)$ (all logical features and neuron-specific parameters travel with their logical entry). Hence $G_\rho^{-1} F G_\rho = G_\pi^{-1} F G_\pi$ on task-local state. This proves one transition, and induction over committed boundaries proves trace invariance.

The condition excludes a lane-specific weight, affinity-dependent numerical mixture, untransported random stream, changed reduction order or hardware arithmetic. It is an allocation specialization for fixed admitted sets, not a proof that all priority policies have the same latency or that a trained router can be removed from every learned model. The archived pool training results retain that model's routing parameters. In the fused joint execution runner the selected set is already capacity-bounded and the lane owner mapping does not enter the numerical operator, so this specialization applies directly.

A lease records task identity, complete group identity and a monotonically increasing generation. Allocation mutates ownership only after capacity and uniqueness checks; release requires the matching task/group/generation and a committed numerical boundary. A stale or duplicate release cannot free a new owner's group. Task-private state is indexed independently and survives release. Fair queue selection and feasible full-group capacity are separate liveness requirements. With heterogeneous groups, the same invariance applies only within an explicitly established equivalence class of execution semantics.

B Experiment Identities and Interchangeable-component Comparisons

Each study reported here has its own weights, run identity and comparison boundary: no checkpoint, accuracy or timing value is substituted across rows, i.e. each row keeps its own run identity, and the following subsections give the archived interchangeable-component protocols.

B.1 Interchangeable LIF components: archived performance comparisons

These runs compare the fused LIF component with CuPy multi-step LIF (SpikingJelly). Projection/readout are common in the model interval; LIF-only intervals exclude them. All speed ratios below divide CuPy-arm time by fused-arm time. The original same-topology CuPy study uses scales $(N, T, d, m) = (16, 24, 4, 24), (64, 96, 64, 24), (128, 192, 64, 24)$ and active fractions 0.25/0.5/1.0 for both methods. Constants beta 0.9, threshold 0.4, subtractive reset and explicit initial state are common. Three repetitions contain 27 model conditions and nine LIF-only conditions. Model speedup medians by run are 3.176, 3.340 and 2.679; the pooled descriptive median is 3.130. This GPU timing regime uses 50 warmups, five rounds of 200 repeats and AB/BA counterbalance. Projection and readout belong to the model view. It excludes backward and DHG service. Runner, CUDA wrapper/kernel, SpikingJelly and CuPy metadata hashes match the archived manifests.

B.2 Interchangeable LIF components: validation and balanced timing

The expanded original protocol freezes five seeds, three scales, widths 24/48/96, three active fractions and two views: 270 cells. Of these, 253 pass the strict gate; the remaining 17 are preserved in the archive without tolerance changes. Twelve have identical spikes with small final-voltage differences; five have a spike disagreement. The gate requires exact spike equality and atol=10^{-6}, rtol=10^{-5} for numerical outputs.

The discrepancy is caused by an implementation-level arithmetic difference. The earlier fused LIF evaluates beta*v+I with FMA disabled; CuPy multi-step LIF (SpikingJelly) evaluates (I-decay*v)+v with its default NVRTC fast-math options. Both express the same real recurrence but can differ at FP32 thresholds. The revised fused backend preserves per-neuron register recurrence and adopts the upstream arithmetic expression and compiler options. It does not modify SpikingJelly, change the input/weight cells or relax the acceptance gate. Every new input and weight hash matches the corresponding original frozen cell.

Expanded matched-arithmetic view	Cells	Pass	Wall ratio	CUDA ratio	Seed spread
Projection–LIF–readout	135	135	1.521	1.568	1.508–1.539
Isolated LIF	135	135	1.874	2.055	1.841–2.191

Table 2: Performance comparison of interchangeable LIF components: fused LIF versus CuPy multi-step LIF (SpikingJelly), RTX 4060 Laptop. Ratios divide the CuPy-arm interval by the fused-arm interval; rows specify the timed computation. All recorded spike, final-membrane and model-output differences are zero. Wall ranges are 1.337–1.579 (model) and 1.768–2.312 (LIF). Every cell is retained, and the per-seed detail of the earlier separate table is carried here. Seed spread is the min–max of the five per-seed medians; each seed forms 27 configuration ratios per view from within-cell medians, and ranges describe configuration spread, not confidence intervals. Per-seed model medians are 1.539/1.512/1.508/1.523/1.526 for seeds 0–4, with configuration ranges 1.403–1.560/1.337–1.579/1.439–1.567/1.383–1.563/1.366–1.557; per-seed LIF medians are 2.191/1.878/1.841/1.853/1.845, with configuration ranges 1.790–2.312/1.770–2.198/1.768–2.188/1.772–2.256/1.782–2.273.

An initial timing run exhibited order rotation (AB/BA reversal cancelled rotation, confounding within-cell order with seed). Its 270 exact numerical agreements remain valid, while Table 2 uses the corrected sibling runner, which keeps all input, weight, arithmetic and gate hashes fixed and directly alternates AB/BA. All 70 pre-existing evidence files are unchanged. Median ratios of within-cell wall means are 1.446× (model) and 1.792× (LIF), distinct from ratios of medians.

The corrected run uses 20 warmups and three rounds of 50 individually synchronized paired calls, with exactly 25 AB and 25 BA pairs per round in every cell. Wall time includes wrapping, tensor allocation/view formation and projection/readout as applicable; compilation is excluded. The GPU is an RTX 4060 Laptop, FP32, TF32 off, torch 2.11+cu128, SpikingJelly 2.0.0rc1 and CuPy 14.2.0. Five seeds are independent units; 135 conditions are configuration coverage, not 135 independent replications. Triton is unavailable in this Windows environment. This is a new implementation variant and timing regime. Its ratios are not combined with the earlier 3.130× result, and passing the finite matrix does not prove equality on every possible FP32 input.

C Internal ablation: Capacity, Packing and Recovery Controls

C.1 Internal ablation: admission-policy and tensor-packing comparisons

The public workflow metadata comes from WfInstances commit `0b74b1b26f4449d29ed7ccec57e57e76452b98a5`. Montage, Epigenomics and Seismology instances have 24/48/96 nodes, five mappings, capacity fractions 0.25/0.5/1.0 and pausable/non-pausable streams. Four strategies give 1,080 completed rows, with no partial allocation or fallback. Checkpoint hashes identify the trained SHD projection/readout used in each run. Service accuracy penalizes unserved tasks; it is not the test accuracy of a newly trained model. The permanent-partition control permits only its reserved task subset and is a mechanism ablation. Shared FCFS gives the same capacity-reuse opportunity. At 25% capacity, FCFS/pool waiting p95 is 16.42/16.21 waves; at 50%, 5.92/5.96. These ordinary-load results do not support a general priority gain.

The original service run packs its admitted set; the arm label it carried is a historical name, not a distinct execution mode (Appendix H). Its later packed RTX 5090 confirmation separately has 270 semantic pairs and 900 timing rows. Predictions, owners and service metrics agree in all pairs; the maximum stored spike-trace difference is 5.9605×10^{-8}. Across six capacity/stream groups, masked/packed wall ratios (ELYSIA internal packing ablation) are 1.026–1.031. The separate internal tensor-packing experiment checks 45 scale/difficulty/seed checkpoints. Its performance rows use each scale's medium/seed-zero checkpoint, 50 warmups and five rounds of 200 repeats; they are not five-seed training means. The large 25% case yields 1.3023× forward and 1.4721× train-step speedup. Waiting-state checks establish that all 48 waiting tasks in the medium 25% case retain their membranes. The repaired active-count CSV only fills an omitted diagnostic; it preserves the original timing measurements.

D INTERNAL ABLATION: PREREQUISITE-RULE COMPARISON

These CPU wave simulations test completion time versus consumed-input coverage on hand-authored graphs. All arms share node-order admission and capacity $C = 8$. Static requires all inputs verified; declared quorum accepts a stated minimum and consumes all currently verified inputs. This changes the consumer contract. The unsafe verification-bypass control (source key `oracle`) waits for parent computation but ignores verification.

Fixed graph and input accounting. The graph has 24 search nodes, six four-input summaries and one conclusion, with durations 3/2/1 waves and one seeded conflicting leaf. Its status becomes verified after an eight-wave delay. The conclusion requires all summaries verified. Block (a) of Table 3 compares $4/4$ and $3/4$ summary rules over seeds 0–19; SD uses denominator 20. The $3/4$ rule reduces mean completion from 16.55 to 12 waves (27.5%), with mean consumed-input coverage 0.9583.

Coverage averages the consumed fraction at summary starts, including reruns. Verified leaves at conclusion completion count even unused inputs: $23.4/24$ is therefore distinct from consumed coverage $23/24$. Slot-time is $\sum_t |\mathrm{running}_t|/(CT)$, an abstract capacity-use fraction. The unsafe arm consumes one conflict and repeats that summary, adding one execution. These metrics do not measure GPU utilization or answer quality.

Separate staggered threshold sweep. Using leaf durations $3 + (\text{leaf id} \bmod 3)$, the same simulator sweeps $q = 1, 2, 3, 4$ over the same seeds. A summary can start before every leaf finishes and consumes every verified input available at admission; coverage may exceed $q/4$. Block (b) of Table 3 records this separate protocol.

Twelve topologies and four conflict counts. The generator varies (P, g, d): leaf count, grouping factor, and terminal stage index. It creates $d - 1$ grouping stages followed by the conclusion. Leaf/grouping/conclusion durations are 3/2/1 waves, with the same eight-wave verification delay. The four levels $c \in \{0, 1, 2, 4\}$ count flawed leaves; their fraction is c/P. There are $12 \times 4 \times 3 \times 20 = 2{,}880$ runs. Here all arms require every parent computation to finish first. Static then requires all n verified inputs; quorum requires $\max(1, n - 1)$, including at the conclusion. Coverage averages consumed fractions over all non-leaf starts, so its denominator differs from the fixed-graph protocol.

For each topology/nonzero-conflict pair, reduction is $100(\overline{T}_S - \overline{T}_Q)/\overline{T}_S$ after averaging seeds. The unweighted mean across 36 such pairs is 22.6615%, with range 3.4749–41.3681%. All 48 quorum cells (960 runs) have zero conflict-consumption violations while accepting reduced coverage. This supports a declared completion/coverage tradeoff; consumers requiring all inputs must retain the all-input rule.

Interpretation and reproduction. These simulators transport verification flags rather than learned numerical artifacts. Verification repair is a delayed status change without a separate capacity charge. The fixed simulator recomputes summary status at completion; it does not retain immutable numerical input snapshots. The topology unsafe arm counts conflict consumption but propagates no conflict and performs no rerun, so its faster times do not establish valid same-result execution. Wave reductions are logical progress counts; GPU timings are reported separately. The learned and full-GPU experiments separately test numerical dataflow and actual recovery costs.

The `evidence_unlock_policy/` package contains unchanged source/JSON, all 144 topology-policy cells, hashes and metric definitions. Its retained CPU audit checks 60 fixed-policy and 2,880 topology runs (162/162 aggregate checks); a new 80-run threshold replay matches all 20 stored means within 10^{-12}. `summarize.py` regenerates tables and repeats that sweep; `-replay-all` additionally checks both other protocols. The records come from the original `simulate` function, not its differing trace helper. No GPU experiment was rerun for this restoration.

(a) Fixed graph (24 search nodes, six summaries, one conclusion), seeds 0–19

Arm	Makespan (waves)	Slot-time	Coverage	Verified (of 24)	Violations
Static $(4/4)$	16.550 ± 2.376	0.6551 ± 0.0920	1.0000 ± 0.0000	24.000 ± 0.000	0 ± 0
Quorum $(3/4)$	12.000 ± 0.000	0.8854 ± 0.0000	0.9583 ± 0.0000	23.400 ± 0.490	0 ± 0
Unsafe bypass	16.550 ± 2.376	0.6705 ± 0.0942	1.0000 ± 0.0000	24.000 ± 0.000	1 ± 0
Node executions	31 ± 0	31 ± 0	32 ± 0	—	—

(b) Staggered threshold sweep, leaf durations $3 + (\text{leaf id} \bmod 3)$*, seeds 0–19*

$q/4$	Waves	Slot-time	Coverage	Verified (of 24)	Violations
1/4	15.05	0.9055	0.8375	23.40	0
2/4	16.00	0.8516	0.9167	23.50	0
3/4	16.10	0.8468	0.9187	23.50	0
4/4	19.40	0.7070	1.0000	24.00	0

(c) Twelve topologies, $c \in \{0, 1, 2, 4\}$ *flawed leaves, 20 seeds per configuration*

(P, g, d)	Nodes	Waves S/Q/U	Q coverage	Violations S/Q/U	Reduction
(16,3,2)	23	14.33/10.81/9.00	0.9357	0/0/1.587	29.41%
(16,4,2)	21	14.33/11.16/9.00	0.9463	0/0/1.400	26.71%
(16,5,2)	21	14.33/11.46/9.00	0.9540	0/0/1.450	24.22%
(24,3,2)	33	16.50/12.61/12.00	0.9449	0/0/1.613	28.78%
(24,4,2)	31	16.50/12.96/12.00	0.9540	0/0/1.513	26.36%
(24,5,2)	30	16.50/13.20/12.00	0.9569	0/0/1.462	24.71%
(24,4,3)	33	18.50/14.96/14.00	0.9642	0/0/1.513	23.70%
(24,6,3)	30	18.50/15.24/14.00	0.9684	0/0/1.375	21.90%
(32,4,2)	41	18.91/15.65/15.00	0.9684	0/0/1.238	21.34%
(32,4,3)	43	20.91/17.65/17.00	0.9741	0/0/1.238	19.42%
(32,8,2)	37	18.91/16.20/15.00	0.9800	0/0/1.062	17.96%
(48,4,3)	64	26.57/25.04/25.00	0.9889	0/0/0.763	7.40%

Table 3: Internal ablation: prerequisite-rule comparison, three protocols in one table. (a) Fixed durations; mean $\pm$ population SD over 20 seeds with SD denominator 20; the completion reduction is $(16.55 - 12)/16.55 = 27.4924\%$, a ratio of seed means; coverage averages the consumed fraction at summary starts including reruns, and the unsafe verification-bypass control consumes one conflict and repeats that summary, adding one execution (31/31/32 = static / quorum / unsafe). (b) Means over seeds 0–19; all thresholds and stored metrics are shown; a violation counts a consumer start with a conflict-flagged input; coverage may exceed $q/4$ because a summary consumes every verified input available at admission. (c) All 12 topologies, S/Q/U = static / quorum / unsafe; waves, coverage and violations average all four conflict counts equally and each cell averages 20 seeds. S/U coverage is 1.0000 and S/Q violations are zero in every cell; reduction averages the three nonzero-conflict cell percentages for each topology.

E LEARNED-GRAPH EVIDENCE: COMPLETE-TASK AND CONSUMED-LINEAGE COMPARISONS

The model, split, capacity, fault and timing protocol were frozen before fitting and GPU measurement. This study has its own weights and run identity; its accuracy is not substituted into any other study.

Learned numeric dependencies. The fine view uses the first original dense-SNN seed (20260820), selected before this experiment's predictions were observed: a trained $700 \to 24$ projection, subtractive LIF and trained $24 \to 20$ readout, on full-length 4 ms binary event sequences. The coarse view groups 700 cochlear channels into 32 adjacent groups and 64 bins of 20 ms; a fixed count/8 projection feeds LIF neurons with $\beta = 1/2, \theta = 1$. Four 16-bin spike-mean blocks give 128 features to a trained ridge readout. The two actual score vectors are concatenated for a trained $40 \to 20$ ridge fusion, followed by a class root. These parents are neither shifted copies nor numerical placeholders. Setting one parent's score to zero changes 1,660/1,174 test predictions for the fine/coarse intervention. These are descriptive frozen-head interventions, not retrained ablations. The shift structure measured elsewhere in this paper is a different object and does not qualify that sentence: the $0/1/2$-bin shift study creates three views of *one* shared producer by shifting its input, behind a single already fitted readout, and its three branches are explicitly not independently trained models (Appendix G.5); the two parents here are separately trained networks with different projections, parameters and readout features (Appendix G.6). How much of the certificate coverage reported anywhere in this paper is attributable to a shift or zero-insertion structure rather than to the certificate's own bounds was not measured, so no such share is claimed here.

The reconstructed seed-20260821 permutation of 8,156 official training examples has 7,340 parent-fit and 816 fusion-fit examples. Split hashes match the original archive. The latter 816 had already selected the fine parent's checkpoint and learning rate: they are reused validation, not a fresh holdout. Both new ridge fits use predeclared alpha 1 and an unpenalized bias. All 2,264 official test examples remain outside fitting and selection, so the test data remain held out. Fine, coarse and fusion correct counts are 1,235, 1,284 and 1,301. The study uses one frozen model.

Two parents never agree everywhere, which is why the disagreement structure matters for the reference question. The parents reach 54.55% and 56.71% and agree on 957 of the 2,264 roots, 133 of those both wrong with the same wrong class; on the other 1,307 they disagree, with the fine view alone right on 411, the coarse view alone on 460, and neither on 436. Every number above is a property of the frozen pair, not of the executor: the two execution arms reproduce the same predictions bitwise (§5.2).

Numerical and execution contract. Both arms time the same complete compound execution: only the LIF numerical implementation differs—fused LIF kernel versus unmodified CuPy multi-step LIF functions—while projection, parent readouts, fusion, dependency, state/lease/recovery control and the timed interval are common. Both backends use FP32/TF32-off with upstream SpikingJelly charge order `(I-(1-beta)*v)+v`, subtractive reset and zero initialization. The fine readout applies its linear map after mean spike aggregation; this changes floating-point order relative to the old runner, so all predictions are reevaluated. Eight inert lanes pad the fine view to a common 32-neuron group. Frozen inputs, weights, source and split hashes accompany `implementation_learned_joint/`.

The identical typed all-parent controller implements the conjunctive dependency and generation-repair subset of the DHG. At most four or eight groups advance 24 bins per dispatch, with deterministic age selection, private persistent state and actual leased-group gather/scatter. This capacity limits simultaneously executing groups, not total resident memory; task-private states and spike histories remain allocated. Different input widths and neuronal parameters dispatch separately under that same limit. A complete, finite, correctly shaped current-generation artifact must commit before fusion can consume its GPU score tensor. ELYSIA uses the register-resident thread-per-neuron fused LIF kernel. The external arm calls unmodified SpikingJelly 2.0.0rc1 multi-step CuPy functions; instrumented probes verify both neuronal parameter settings use those entry points. Projection, readout, fusion, state and all control are common.

A missing-artifact trial drops the first utterance's complete fine-parent output before commit. A timeout trial rejects that parent's first chunk. Each happens once per episode. Recovery retires its lease, rejects an old generation write and recomputes that parent from zero while unrelated artifacts remain valid. These are synchronous injected failures, not actual deadline misses or arbitrary concurrent stale GPU writes. CPU controls reject incomplete, stale, malformed and nonfinite artifacts and invalid lease updates.

Complete quality and paired timing. Every capacity/condition covers 71 episodes of up to 32 test utterances: 852 episodes across both backends. All 27,168 repeated quality roots complete; 568 repeated failures recover and 568 stale writes are rejected; there are 71 distinct fault-target utterances, repeated across capacities, conditions and backends. The 426 quality pairs and 120 timing pairs have identical artifact/state hashes, spikes, predictions and dispatch traces, with zero numerical differences. CPU and GPU predictions agree for the full official test split.

Each timing cell uses the first 32 official test utterances, two warmups per arm and 20 paired repeats, exactly 10 AB and 10 BA. The synchronized wall interval starts before input upload and controller/state initialization and ends after final prediction download. It includes projection, LIF, readout, fusion, group leasing, artifact checks, host control and repair. Disk/event decoding, fitting, weight upload, context initialization and JIT are excluded; initial setup is 1.731 s. Hashing and file writing follow timing. CUDA events also include stream idle time during host work and are not pure kernel costs. All individual reversals are retained. Ratios of within-cell means have median 1.186×, compared with 1.201× for medians. All six cells of Table 6 favor the fused LIF, including both recovery conditions. PyTorch-tracked incremental peak allocation is 21.86–22.39 MiB in both arms; this excludes allocations outside PyTorch and is not whole-process GPU memory.

The six conditions share a single frozen model and fixed timed inputs; 20 repeats per condition do not create independent models. GPU clocks are unlocked, and host overhead is part of this bounded experiment. Our timing worker for the SpikingJelly arm completed and released its lock before this run. The first attempt stopped before GPU measurement due to an older package on the import path; its record is preserved. The canonical run explicitly selects the verified SpikingJelly installation without changing any model or gate. Raw per-instance predictions, paired records, runtime snapshots, prepared inputs and frozen weights are supplied with a relocatable runner.

E.1 Internal ablation: consumed-lineage repair policies

This separate experiment compares *local lineage repair* with *affected-root restart (ELYSIA control)*. Both use the same fused LIF component, learned graph, tensor arithmetic, packing and leased-group capacity. Restart is this paper's own same-kernel control, ELYSIA-P: it shares the kernel with the arm it is compared against, so the only thing that differs is whether dependency machinery is carried, and that is what the comparison costs. It evaluates recovery policy; no external component changes.

A multilevel learned numerical graph. The frozen fine SNN A and coarse SNN/readout B from Appendix E publish 20 scores each. The new F concatenates these 40 scores, standardizes them with training-fit mean/SD, and uses a trained $40 \to 32$ projection and bias followed by $0.5 + 0.5\tanh$. These fixed currents drive a 32-neuron subtract-reset LIF, $\beta = 0.75$, $\theta = 1$, for 24 steps. F publishes mean spike rates; trained readout H maps them to 20 class scores. Thus $A + B \to F \to H$ has two downstream consumption levels, including a trained nonlinear SNN. The earlier linear fusion weights are retained as provenance but are not executed. F advances in three 8-step chunks through the same private membrane and group-lease path as the source SNNs, which use 24-step chunks. Every numerical node contributes to prediction.

The 1,972 new projection/readout parameters are trained jointly on 816 official training examples, using hard forward spikes, a surrogate derivative $1/(1 + 5|z|)^2$ in the backward pass, and detached resets. The fixed protocol uses seed 20260914, 120 Adam epochs, batch 136, learning rate 0.01 and weight decay 0.0001; it takes the final epoch without test selection. These 816 examples previously selected the fine-parent checkpoint and learning rate; they are reused for fitting and are not presented as a fresh holdout. The coarse readout was fit on the disjoint 7,340 training examples. The trainer loads training data only. Fit accuracy on this partition is 816/816. The frozen model scores 1,530/2,264 (67.58%) on the official test set. This one-model result has its own weights and

protocol; it does not replace the 57.46% linear fusion experiment. No training-seed uncertainty or accuracy superiority is inferred across those studies.

Late correction after descendant consumption. For row zero of a batch, one condition withholds the first 24 fine bins of A; the other withholds the first 24 coarse bins of B. The sibling stays complete. Only after all provisional roots finish does the controller restore the selected input bytes and advance their version. Checksums verify the change. These are delayed/corrected view representations. The old source has already been consumed by F, and F by H. Recursive version validation rejects the obsolete root immediately, before artifact removal; stale commit tickets for source, fusion and readout are rejected as well.

Local repair resets exactly the changed source, F, and H: three nodes. The control resets both sources, F, and H of the affected utterance: four nodes. Both retain all other utterances. In a 32-utterance batch, local repair retains 125 artifacts and the control 124. Reset traces verify erasure of the consumed binding, cached currents, completed cursor, spike history and membrane; replay starts from zero under a new generation. All final current artifacts match a fresh complete reference. The control restarts only the affected request, not the entire batch.

The complete quality sweep has 71 batches, both source corrections, capacities 4/8 and both policies: 568 fresh/recovery comparisons and 18,112 current-root checks. It corrects 71 distinct utterance targets per source, while checking all 2,264 roots; repeated capacities do not create new targets. Scores, spikes and final membranes agree bitwise at every node. A correctness-only negative control retaining obsolete state changes scores in 71/71 targets for each source, and changes classes in 15 fine-view and 60 coarse-view targets. Version validation rejects obsolete roots even when their classes happen to remain unchanged.

Complete-time comparison of internal repair policies. Five predetermined disjoint batches use official indices $[0, 32)$, $[448, 480)$, $[896, 928)$, $[1344, 1376)$ and $[1792, 1824)$. Each capacity/source/batch cell uses two warmups per arm and ten AB/BA pairs, ten AB and ten BA. The 200 pairs give 400 timed-arm fresh-reference checks, all bitwise equal; the five batches cover 160 utterances. There are no outcome-based exclusions. The device is an RTX 4060 Laptop GPU, with PyTorch 2.11.0+cu128, CUDA 12.8, CuPy 14.2.0 and TF32 disabled. Projection/readout use fixed per-node tensor shapes so different packed widths do not change reduction arithmetic; only the pointwise LIF population is packed.

Synchronized wall timing includes input clones/checksums, uploads, controller/ state creation, provisional graph completion, corrected-view upload, version update, invalidation, reset/replay, root validation and final score download. Complete wall time therefore includes initial execution, correction and replay. It excludes model fitting/loading, event unpacking, weight upload, context/JIT setup and post-run audit serialization. CUDA event spans include host-induced idle time and are not isolated kernel timings. The first arm's outputs can remain allocated during the second; order is balanced and memory records are descriptive. Table 4 reports every cell.

At the twenty cells of Table 4 the lineage machinery's replay saving is real and its wall-clock saving is not established: nine of the twenty ratios sit at or below 1.0, and the rest sit close above it, so this table does not separate the two policies. Separating them would need a regime with larger subtrees per branch or more affected roots per revision, so that one revision destroys more than a fixed cost of bookkeeping.

For fine corrections at capacities 4/8, the median of the five ratios of arm medians is 1.034/0.997, with ranges 0.922–1.094/0.975–1.107. For coarse corrections it is 1.003/0.918, with ranges 0.940–1.048/0.858–1.240. The corresponding medians of pairwise-ratio medians are 0.953/1.039 and 1.030/0.997; these are distinct descriptive statistics, and the sign of the wall-clock difference is not stable at five batches. Retaining one sibling saves 21.3–26.9% of replay neuron-steps for fine corrections and 63.0–70.8% for coarse corrections across these batches. Initial computation of all 32 provisional roots is common to both arms, so that saving need not dominate complete execution time. Using the recorded complete-interval neuron-step count as denominator, the savings are 0.714–0.753% for fine corrections and 1.727–2.419% for coarse corrections. This includes initial work shared by both policies; it is a work-count interpretation, not a decomposition or prediction of wall time. Timing repeats are not independent trained models or an input-population confidence interval.

View	Groups	Start	Local ms	Restart ms	Ratio	Paired ratio	Replay saved %
A	4	0	304.05	314.33	1.034	0.953	23.6
A	4	448	368.94	388.02	1.052	1.022	24.7
A	4	896	495.28	458.86	0.926	0.910	25.0
A	4	1344	490.19	451.92	0.922	0.937	26.9
A	4	1792	344.75	377.28	1.094	1.094	21.3
A	8	0	317.73	315.36	0.993	1.039	23.6
A	8	448	390.50	380.79	0.975	1.005	24.7
A	8	896	332.48	367.34	1.105	1.092	25.0
A	8	1344	312.56	311.53	0.997	0.932	26.9
A	8	1792	292.11	323.35	1.107	1.089	21.3
B	4	0	395.77	371.99	0.940	0.975	67.5
B	4	448	391.37	374.45	0.957	1.003	66.0
B	4	896	440.23	461.35	1.048	1.189	65.6
B	4	1344	412.71	414.06	1.003	1.036	63.0
B	4	1792	330.69	339.34	1.026	1.030	70.8
B	8	0	297.40	368.66	1.240	1.178	67.5
B	8	448	407.25	349.39	0.858	0.971	66.0
B	8	896	417.69	383.63	0.918	0.997	65.6
B	8	1344	364.72	340.88	0.935	1.003	63.0
B	8	1792	307.61	278.28	0.905	0.905	70.8

Table 4: Internal ablation: local lineage repair versus affected-root restart (ELYSIA control), the consumed-lineage graph $A + B \rightarrow F \rightarrow H$ of this appendix. View A/B denotes a corrected fine/coarse source; Start identifies a 32-utterance batch. Affected-root restart is the ELYSIA-P behaviour: with no recorded lineage the only safe response to a revision is to redo the whole graph. ELYSIA-P is this paper's own same-kernel control — the same fused kernel, graph and capacity as the repair arm, with the dependency machinery removed — so the two columns separate that machinery's saving from the kernel's. Times are arm medians for the complete interval. Ratio is restart median/local median; Paired ratio is the median of ten pairwise restart/local ratios. Values above one favor local repair. Replay saving counts neuron-steps after correction, not the whole interval.

Scope and preserved protocol history. This is serialized application recovery after completed descendants have consumed an input, with source revision ordered against commits/consumption. It is separate from certificate-directed sparse numerical repair and from arbitrary in-flight GPU races. Functional tests additionally exercise partially consumed descendant resets and diamond closure. Active groups bound temporary leased execution capacity, not total memory; inputs, artifacts and private states remain stored. The four-node learned DAG and 31 unaffected requests do not establish arbitrary-topology scaling. Mixed class/full-spike consumers are permitted by the architecture's per-request interface but are not exercised by this all-full-artifact learned graph or by the separate all-class studies.

Before test evaluation, the initially proposed full-batch restart was strengthened to affected-root restart, and both source corrections were fixed. Draft and revised protocols, source snapshots and timestamps remain archived. One preflight stopped at an ownership lock; a second stopped before test loading because a separate-op CPU reference differed from fast-math CUDA by 4.768×10^{-7} in membrane, with equal spikes. The actual unmodified CuPy LIF component (SpikingJelly) matched fused LIF bitwise in the synthetic setup probe. The corrected probe was frozen before any test evaluation; model, experimental kernel, exact gate and timing plan were unchanged. These preflight records have no quality/timing rows, and are preserved alongside the complete successful run. The setup probe is a component correctness check, not a performance result.

The portable `implementation_multilevel_learned/` directory includes all training/prepared inputs, weights, source snapshots, 284 quality records, 200 timing pairs, full test scores, consumed-lineage traces, protocol history, analysis, source hashes and commands. The frozen model SHA256 begins `4a13a24a93d10efb54e0dc685a47ce3a5`; the complete identity is in the manifest.

F Full Accounts of the Main-Text Measurements

The nine-page main text states each result once, at the length the page budget allows. This appendix carries those results in full: the measurements, sweeps, per-arm settings, proofs and validations behind them. Nothing here is a new experiment, and every number here measures a result the main text states or points to; where an appendix carries a finer breakdown, the extra figures are that result's own components rather than new claims.

F.1 Numerical release policy: the full account

This appendix carries, in full, the material that §5.1 summarizes. Nothing here is a new experiment; every number is the one the main text cites, with the surrounding measurements that the nine-page main text cannot hold.

The three costs, and which conditions each guarantee needs. The cost of the guarantee is not one number, because the guarantee is available on contracts of very different quality. Table 5 separates them. Inside the exact-dyadic conditions the same guarantee that the main text quotes at $553\times$ costs 6–15% when the release decision is not taken early, and 2.23–$7.8\times$ when it is. Only the fourth row leaves the contract: the 2–3 orders of magnitude in the batched column are what that model costs, because no fixed dyadic denominator represents β^t.

What you want	Ratio, unbatched	Ratio, batched	What it gives
Plain forward, no certificate	$1\times$	$1\times$	no guarantee
Exact-trace certification	$1.06\times$	$1.15\times$	every class released, 0 wrong
Exact-dyadic scan, early release	$2.23\times$	$7.8\times$	same guarantee, released at round 3 of 64
Enclosure, either side of those conditions	$362\times$	$553\times$	same guarantee where the scan cannot run

Table 5: The cost of the guarantee, by the conditions the model satisfies, each ratio named with the model, the device and the protocol it was measured on. Rows one to three are the same $T = 64$, $\beta = 1/2$ model inside the exact-dyadic conditions of Appendix G.8, measured on an RTX PRO 6000 Blackwell under one vote-inclusive protocol (Table 32), and the batched column charges the plurality vote to every arm alike. The unbatched column is the stable one across hosts: the $2.23\times$ of this device reproduces the $2.08\times$ published on the RTX 4060 Laptop, whose per-request boundary and batch protocol differ from this one, and the batched column is the one that moves with the device, $7.8\times$ here against the published $4.82\times$ on the RTX 4060 Laptop boundary of Table 32. Row four is outside those conditions, and it is recorded rather than remeasured here: its $553\times$ is the enclosure on the 512-neuron model of Table 34, whose leak $\beta = 19/20$ no fixed denominator 2^k represents, at batch 2,264 against this paper's own fastest no-guarantee arm, itself at a different accuracy (0.751325 against the certificate's 0.769435), timed on one RTX 5090 (Appendix F.1); its $362\times$ is that enclosure's per-request ratio against this paper's own fused trace on the 57.51% model of Table 14, a different pipeline. Every certified row keeps the released class equal to the reference class, so none of them changes an answer, and the two paths agree on that while differing in cost by two orders of magnitude.

Every method on one frozen pipeline. We first ask what early release costs in answers. Figure 2 puts every method we could configure on that one frozen pipeline, as work executed against the number of test classes whose answer differs from the full trace's, at the best setting of each method's own knob (Table 17). Every prior method gives speed with changed classes, and no swept setting of any of them reaches the certificate's coverage at zero changed classes: the nearest calibrated alternative does reach zero, but only by releasing 1.46% of the split (Table 13), so what separates the arms is coverage at zero, not zero itself. The per-sample oracle, which is handed every root's final answer and is not deployable, shows the headroom is real — it exits at a mean of 29.9 timesteps of 100 while changing nothing. Reaching zero at low coverage is not the difficult part; doing it while releasing four fifths of the split is.

The three senses of the certification round. The certificate's certification round needs its three senses separated, because they are different quantities and only one of them is saved work. *Exact tail*: given the exact reference trace, one sound causal enclosure pass pins every coordinate, which is what the certification round indexes. *Unknown tail*: deciding the class *without* evaluating the rest of the stream, which is the latency question a deployer asks. Under the sequential-sweep rule this paper ships and the shipped readout it releases 0 of 2,264 roots before the last step; the same sequential-sweep rule closes 499 of 2,264 proofs within its 60 rounds in the exact-tail sense; and the hull rule releases 2,264 of 2,264 at a median round of 12 on the same deployed recurrence (Table 1, Appendix I.2). *Parallel rounds*: the round-by-round iteration, whose *agreement channel* contributes exactly the advancing prefix and nothing beyond it, so the Theorem-3 mechanism adds no release outside $t < K$. The shipped certified path computes the exact-reference serial LIF for the whole batch before any round runs, in both its full and its early-release modes, because the prefix is pinned to it. It therefore executes the full trace and then the enclosure rounds and the exact-integer readout on top of it, and the 553× in Table 5 is what that costs on the non-dyadic model, against this paper's own fastest no-guarantee arm. Its certification round is when the proof closes, not work skipped. We therefore plot it in Figure 2 at the work it actually executes, with the full trace, and mark it as the only arm there that also carries the proof. That is not merely an implementation outcome: with exact arithmetic and unbounded parallelism the release decision costs depth $K\lceil \log_2 T \rceil + \lceil \log_2 T \rceil$, which *exceeds* the serial trace depth at the paper's own scales — $90 > 64$ at $T{=}64, K{=}14$ and $126 > 100$ at $T{=}100, K{=}17$ — and wins only from $T{=}256, K{=}17$ on ($144 < 256$). The bound describes the ideal parallel route; our shipped sequential-sweep rule is sequential and does not take it. The hull rule does not remove it either, because the hull rule changes the constant rather than the shape, and its larger $K = 36$ makes the bound worse ($259 > 100$). The crossover is measured, and it follows from the arithmetic. Sweeps, per-arm knob settings, the three senses measured separately and the certificate port's validation are in Appendix G.9. Inside those conditions the guarantee is nearly free, and Table 5 shows where it stops being so: for a consumer whose model admits no exact dyadic representation, the enclosure is what stands between that model and no guarantee at all, and because a downstream consumer's cost is added identically to every arm, no amount of downstream work gives that guarantee more cheaply. On the external method the guarantee's cost is 10.5–15× per request (173–226× batched) at the one accuracy where both were run (0.769435) — a method that changes classes everywhere it exits and reaches that accuracy only where it does not, while the certificate changes none.

What the certificate saves, and its second implementation. A fixed encoder and a learned quantized ridge readout permit exact evaluation on all 8,156 training and 2,264 test examples, and class certificates preserve every reference prediction. On the single-view full test, 1,896 samples (83.75%) certify before all spikes (Table 14); the mean certificate evaluations of that one-view class release are $8{,}557/2{,}264 = 3.780$ per root, against 4.160 for the three-view class-release arm (Table 12). The 5.874 sometimes quoted beside it belongs to a different arm: it is the mean for the three-view *full-spike* arm, or 13,299 producer calls over the same 2,264 roots. The two numbers therefore come from different arms and different view counts, not from one single-view measurement. In the three-view shared-producer study, class release cuts producer maps from 13,299 to 9,418 (29.18%) and releases 1,662 of 2,264 roots at matching wall time (2.669 vs 2.694 ms; Appendix G.10). A per-request measurement costs it: a fused full-trace request takes 0.138 ms (median of 128), and a certified class release 49.9 ms in FP32 and 49.8 ms in FP64 at round 3 with zero wrong (Table 26) — the outward-rounded FP32/FP64 enclosure, which keeps the guarantee on models that lie outside the exact-dyadic conditions. The exact-dyadic class scan is a different and cheaper implementation of the same guarantee: 2.08× the fused full trace per request (2.05× at three views) and 4.82× batched, and 1.90× for a whole split issued as single requests, breaking even at 0.149 ms against the consumer's own 0.148 ms; the serving loop measures the enclosure at 1.6–1.9× the fused arm (Table 32, Appendix I.1). The 4.82× batched figure is a Blackwell device-event ratio on the archived RTX 4060 Laptop boundary: Table 32 carries the device, the timer placement and the same-session rerun, and Table 5 gives the vote-inclusive boundary on the Blackwell, where the same comparison reads 7.8×. Both variants change no class. The exact-dyadic pair is the one where the guarantee and the cost point the same way, and what that cost is spent on is release: the scan costs 2.08× a fused trace that has been truncated at $T' = 82$, the point chosen on the validation split and fixed before the test run. The truncated trace changes 150 classes; the scan changes none. The nearest calibrated alternative reaches 1.46% early release against 83.75–86.31% for the certificate, and outward-rounded arithmetic, the N-MNIST prefix and a leak sweep to $\beta = 0.9375$ all keep its zero-wrong bound (Ap-

pendices G.7, G.8). It also runs on a trained layer inside the learned compound graph and on a 512-neuron model whose leak leaves the dyadic conditions (Appendices I.3, I.7, G–G.11).

What bounds the guarantee. Three further experiments bound what that guarantee is and what it costs. Proposition 8 shows the zero-wrong column is forced by the margin test rather than earned by the model: it is a pointwise statement about agreement with the reference computation, proved without reference to weights, accuracy or data distribution, and confirmed on a population whose model accuracy is 0.00 by construction. Proposition 9 shows the abstention is optimal: at budget K the certificate emits the coarsest release set that is sound for every datum, so no procedure releases more without giving up soundness. The guarantee survives depth: a two-layer chain composes with no new hypothesis beyond higher-precision hand-off and one outward rounding, releasing $2{,}201/2{,}264$ (97.22%) with zero wrong and zero FP64 enclosure violations, and a three-layer chain matches it, while propagating an interval extension over the admissible parent set instead is sound and releases 0 of 2,264. It also survives model quality: on a stronger 512-neuron model at 0.8454 test accuracy ($1{,}914/2{,}264$, against 0.7694 here) the same certificate releases $2{,}264/2{,}264$ at median round 1 of a 256-round budget with zero wrong releases — and the zero-wrong property did not improve with the model, because it does not depend on the model. Appendix G.9 gives all four, with the one counterexample we found: a statistical rule that beats the certificate on a degenerate single-coordinate readout.

The two release rules, side by side. §5.1 reports the comparison of Table 1 in the main text; this paragraph carries the rest of it. Both rules run on the deployed 512-neuron recurrence, update the same sound box once per round, and are measured in one exclusive GPU block with 60.0 s of confirmed peer-free settling before any clock starts. Both pass the same gates first: the deployed FP32 accuracy equals the float64 reference at $1{,}742/2{,}264 = 0.769434628975265$, with 0 class and 0 spike disagreements over 115,916,800 coordinates. The sequential-sweep rule runs its 60 rounds at 6.73 s per round, so its 60 rounds cost $60 \times 6.73 = 403.8$ s of round budget; the hull rule runs 36 rounds at 8.84 s per round, so its 36 rounds cost $36 \times 8.84 = 318.2$ s. The hull rule therefore pays $1.31\times$ per round and reaches full coverage on about a fifth less round budget, because it needs fewer rounds to release more. The hull rule's own certification round is 12 at the median, 13.51 at the mean and 20 at the ninetieth percentile, against our sequential-sweep rule's 54.0, 52.33 and 59.0; under the hull rule no class is left unreleased, where the sequential-sweep rule still holds 1,765 of 2,264 at round 60. Both rules hold 0 wrong pins, 0 wrong released classes and 0 lower or upper enclosure violations over every round they run, so the difference between them is coverage and round count rather than soundness. The two rules visit coordinates in different orders, so the table gives each rule's own release set and its own round count.

Three further measurements in this appendix belong to the same comparison and are the ones the main text points to. The concentrated-readout study releases all 2,264 roots strictly before $T = 100$ from $k = 12$ on, with zero wrong releases at every setting tried, and at matched accuracy it needs $1.33\times$ less trace work than a reset-to-zero truncation. The three-layer chain matches the two-layer chain's $2{,}201/2{,}264$ (97.22%) exactly. And the counterexample is a statistical rule that beats the certificate on a degenerate single-coordinate readout, which is why it is reported rather than suppressed. The external early-exit comparison is costed in Appendix G.9: 173–226$\times$ batched and 10.5–$15.1\times$ per request at the one accuracy where both arms ran (0.769435), a setting at which the external method never exits.

F.2 Certified backend: the full account

Proof sketch of Theorem 6 (asynchronous selected-window bounds). Every old published bit still bounds the same reference bit, so coordinatewise mixing retains soundness. Antitonicity bounds each proposal; monotone merge preserves it. Coordinate zero resolves when updated. Once its predecessors resolve, freshness and continued updates eventually give each next coordinate its exact inputs. Causal induction proves finite-event convergence.

Window summaries. For a window $[a, b)$, the candidate-dependent affine summary $(\beta^{b-a}, \sum_{j=a}^{b-1} \beta^{b-1-j}(I_j - \theta q_j))$ transports its incoming post-reset state. The invalidation rule and the cache it uses are specified in Appendix A.6.

F.3 COMPLETE-TASK COMPONENT TIMING

The fused LIF and the CuPy multi-step LIF of SpikingJelly are timed through prediction download on the learned graph of §5.2, with control, weights, packing and recovery fixed.

Groups	Condition	Fused LIF	CuPy multi-step LIF (SpikingJelly)	Ratio
4	Clean	108.28	131.38	1.213
4	Missing artifact	108.51	131.35	1.211
4	Timeout	107.12	131.11	1.224
8	Clean	86.41	97.92	1.133
8	Missing artifact	87.03	103.75	1.192
8	Timeout	81.24	95.38	1.174

Table 6: Complete-task median wall time in ms on the RTX 4060 Laptop, for 32 fixed utterance roots and 20 runs per configuration (10 runs in each order). Only the named LIF component changes; graph/lease control, frozen weights and recovery are common. Time includes input upload, initialization, projection/LIF/readouts, fusion, commits and final prediction download. The ratio is over the complete task rather than the LIF kernel alone; Appendix B expands the LIF-only comparison.

Replacing this component gives a median complete-task wall ratio of 1.201× over six combinations. All six favor the fused LIF, including both recovery conditions.

F.4 COMPOUND EXECUTION: THE FULL ACCOUNT

This appendix carries, in full, the material that §3 summarizes.

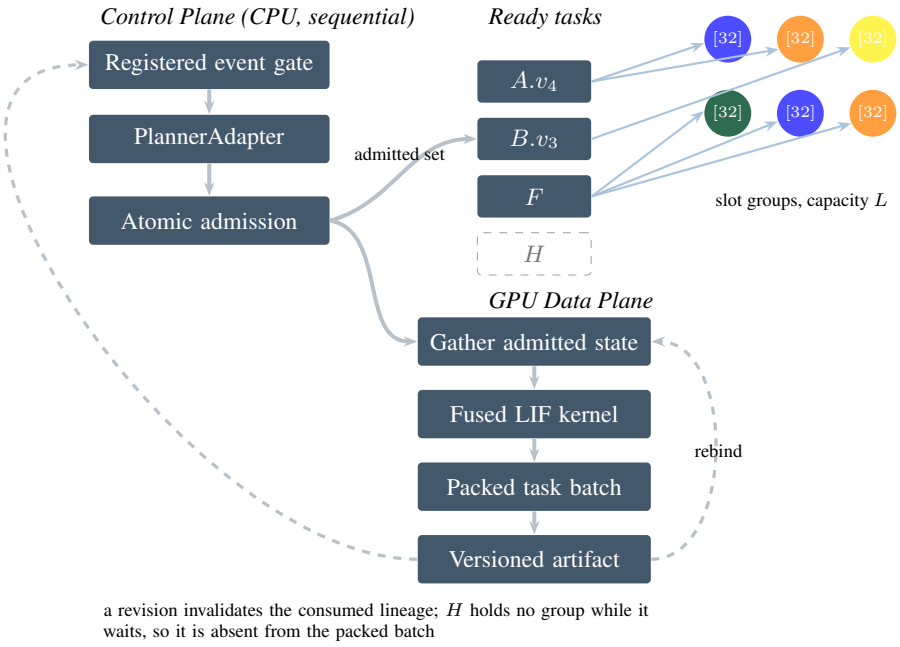


Figure 5: Control plane and data plane. The CPU control plane admits eligible nodes and fixes each node's input, parameter and parent-version identity. Admission is also where the competition on the right is settled: an active task is granted whole neuron groups up to the capacity L, so only admitted tasks reach a group, and a waiting task is absent from the packed batch. The GPU data plane then gathers only the admitted state, runs the fused kernel and publishes a versioned artifact. A revision invalidates the consumed lineage and rebinds the affected descendants (Appendix H).

Running example. The main text fixes one request as its running example (Figure 6). A user submits an SHD utterance to the learned graph $A + B \to F \to H$. Tasks A and B each consume

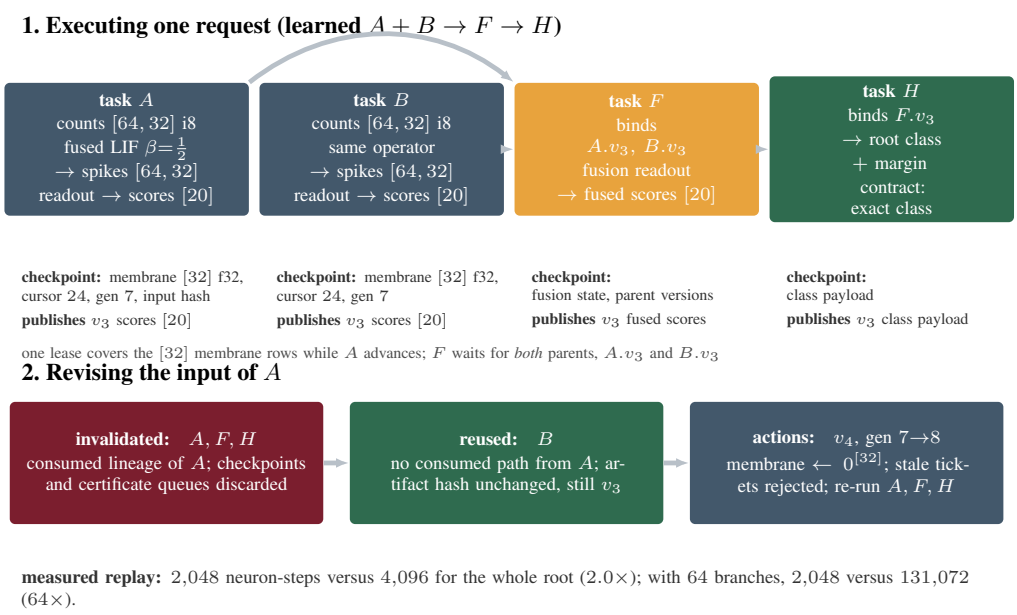


Figure 6: Running example. One compound SHD request over the learned $A + B \to F \to H$ graph, in the vocabulary of §1. It shows the tensors each subtask actually moves, the contents of its checkpoint at a safe boundary — a point at which all resumable state has been written down — and the actions after the input of A is revised. Shapes and values come from the implemented pipeline. An input sample is binned into 64 timesteps and 32 input channels, so one sample is a 64×32 grid of counts; each subtask holds 32 neurons, which is exactly one neuron group; $\beta = 1/2$ is the membrane decay and $\theta = 1$ the firing threshold; and the readout is a frozen exact-integer map from membrane state to the 20 SHD classes, so no rounding enters the class decision. Repair and full re-execution produce bitwise identical root artifacts in every timed revision (864 revisions, 0 mismatches). Those revision counts and the replayed-work figures are measured on the synthetic width sweep of Appendix H, not on this learned request.

one window's event counts $[64, 32]$, task F fuses their payloads, and task H publishes the root class. Task A runs the fused LIF and a frozen int64 readout to a score vector $[20]$. Its checkpoint at a safe boundary holds the membrane $[32]$ in FP32, the input cursor, the partial readout and the input/parameter hashes.

When a node is eligible. Each node is eligible exactly when it is pending, its stage predicates hold, and every required parent version is valid:

$$\text{ready}(v) = \mathbf{1}[v \text{ pending}]\mathbf{1}[\text{stage/predicates hold}] \prod_{p \in P(v)} \mathbf{1}[\text{required artifact version of } p \text{ is valid}],$$

The runtime therefore never binds missing, conflicted or untrusted inputs into a child.

What admission grants. The sets admission chooses are $\mathcal{A}(i)$ with $|\mathcal{A}(i)| = m_i$ for each task's required group size m_i, and $\sum_i m_i \le L$.

Liveness and the scope of a conflict. *Liveness* — every live subtask eventually finishing — requires eventually stable source versions, feasible capacity, fair continued service and eventual validation/solver termination. A conflict's data effects follow consumed edges, while resource competition can affect unrelated completion times.

The flat device time behind the packing split. On the RTX PRO 6000 the fused kernel's device time is flat at 14.6–14.8 μs from batch 1 to 128, so the saving is in executed work rather than in kernel device time.

Advance, publish and the state a task owns. A consumer can start once its artifact contract is proved, while the producer continues for full-trace consumers and retires only when all live requests are satisfied. Task i owns logical state S_i: membrane values, input cursor, accumulated readout, parameter/input identities and backend-specific progress. A temporal computation also stores its bounds, boundary summaries and pending update versions. A safe boundary commits all of it before a lease is released. Slots provide temporary computation capacity; state storage scales with live tasks.

What the matched-kernel comparison holds fixed. The comparison of §5.4 fixes inputs, weights, state, reset, FP32 arithmetic, active sets, graph, control, packing and recovery, so that the two arms differ only in the numerical path. The six exact-dyadic conditions, as §5.1 states them, are $\beta = 2^{-j}$ for an integer j; currents that are integer multiples of one common unit 2^{-3}; $\theta = 1$; $T = 64$; the subtract-θ reset; and an exact-integer readout. A model that meets all six runs the exact-dyadic scan; a model that misses any of them is routed to the enclosure, which is why the 512-neuron $\beta = 19/20$ model cannot use the scan and cannot have the enclosure withdrawn. The eligible-node rule of §3 is the one stated above: a node is eligible exactly when it is pending, its stage predicates hold and every required parent version is valid, which is why the runtime never binds missing, conflicted or untrusted inputs into a child. Validation is the pair of §3: a validator proves a class or spike contract, while a schema or checksum validator establishes only the declared property rather than semantic truth.

Recovery over consumed lineage. An emitted artifact carries a producer/version, a payload and a fulfilled contract. A timeout or an evidence conflict suspends the affected consumption, and recovery applies a checked acyclic graph patch, invalidates the affected consumed lineage and resumes from valid inputs, so unaffected branches need not be rerun. The invalidation rule and the cache it uses are specified in Appendix A.6.

Proof sketches of the two architecture propositions. *Homogeneous lease invariance (Prop. 1).* Allocation merely permutes the gathered logical entries. Slot-independent transitions commute with this permutation, and inverse scatter restores the same task-indexed result. Induction over safe boundaries proves the claim.

Compositional reference preservation (Prop. 2). Sources bind identical external inputs. If all selected parents match, the child binds identical inputs and its local contract yields an identical artifact. Topological induction completes the proof. A recovery patch creates a new versioned reference to which the same argument applies.

Checkpointing under revision. A complete checkpoint resumes under the same valid input/parameter binding and numerical execution contract (Appendix A.6).

When the unknown tail is not empty. The 0 of 2,264 above is a property of the shipped readout, not of the mechanism. If the readout is concentrated on the first k timesteps then the un-evaluated tail carries exactly zero weight, so a release at round k is a genuine decision taken without evaluating the rest of the stream, and the certificate still proves that the released class equals the full trace's. On the subtract-θ pipeline of §5.1, with a 12-bit quantized concentrated readout trained on `shd_train` only and evaluated on the official 2,264-sample test split, all 2,264 roots release strictly before $T = 100$ from $k = 12$ on, with zero wrong releases at every setting tried. The hard-prefix window gives k=64: 0.7482 at median certification round 53 (max 63); k=80: 0.7805; k=100: 0.7809, above the shipped readout's own 0.7694 on the same frozen trace. Holding the recurrence fixed inside each comparison, the smallest reset-to-zero truncation that reaches the concentrated readout's accuracy needs $T' = 85$ steps at k=64, so at matched accuracy the guaranteed path needs 1.33× less trace work than the unguaranteed one; at $k \geq 80$ the concentrated readout exceeds the reset-to-zero model's own full-trace accuracy, so no matched comparison exists there. Three limits bound this. The recurrence is subtract-θ, not the deployed reset-to-zero; the matched comparison is between two different models rather than two settings of one model; and the reduction is in trace work counted in rounds — the exclusive wall-clock block was not measured, so no latency claim is made, and whether the crossover survives the class scan's constant factor is open.

F.5 SERVING THROUGHPUT: THE FULL ACCOUNT

One serving loop drives every arm on identical SHD streams, at capacities 4 and 16 under FCFS or EDF (first-come-first-served, or earliest-deadline-first). The fused kernel raises end-to-end throughput 2.25–3.61× over the same loop driving the package's step-wise LIF on three GPUs at the 24-step windows of Table 22, a margin that grows with window length (Appendix I.5); the top of that per-device range, 3.61×, is the RTX 4060 Laptop's. The gain is dispatch count rather than arithmetic work (Appendix I.5). That margin belongs to the kernel, so the architecture is measured with both executors on one shared kernel — ELYSIA-P is our own stripped prototype, held fixed as the same-kernel control; that is the setting which isolates the cost of dependency bookkeeping: over 18 independently generated streams ELYSIA retains 0.479 $[0.432, 0.535]$ of ELYSIA-P's throughput at capacity 4 and 0.650 $[0.587, 0.694]$ at capacity 16, measured on the one workload that does not need that machinery. A dependent stream cannot use ELYSIA-P, having no dependency graph, checkpoints or contracts to carry the reference computation across suspension and revision. We are not aware of an independently authored compound-SNN executor to compare against, so this axis rests on capability. The per-stream detail, the admission rewrite that raises the ratio to 0.55 and the finding that no pooled figure is reproducible across streams are in Appendix H and Appendix I.8.

What each comparison measures. The comparison above is internal: ELYSIA against ELYSIA-P, both written here. We found no independently authored compound-SNN executor to run against, so this axis reports capability rather than speed. On the release-policy axis the external partners are the SEENN-I, SEENN-II and SpikeCP policies re-implemented from their papers in Appendix G.9; no certified release policy from another group was run from a published checkpoint, so the finding that the certificate is the only arm changing no class is a statement about those re-implementations.

G EXPERIMENTAL DETAIL AND EVIDENCE AUDIT

G.1 SCOPE OF EVERY CLAIM

Table 7 maps each theorem, certificate study and execution study to the model, arithmetic, output and execution assumptions it needs, so the supported scope can be read in one place. Anything not in a row of that table is not claimed: in particular the theorems are stated for a fixed input trace and non-recurrent candidate-independent input, the certificate studies inherit those conditions, and the execution studies measure the FP32 fast path whose spikes we verify against the float reference and—after the tie-rule fix described in Appendix G.10—against the exact dyadic reference.

Table 7: What each result assumes. Every theorem is stated for a fixed input trace, non-recurrent candidate-independent input and sound arithmetic; the certificate studies inherit the same contract, and the execution studies use the FP32 fast path whose spikes we verify against the float reference and, after the tie-rule fix, against the exact dyadic reference. Items outside these rows are not claimed.

Result	Model scope	Arithmetic	Output / execution assumptions
Thm. 3 (sandwich, prefix exactness)	One non-recurrent LIF layer, fixed input trace, $\beta < 1$, $\theta = 1$, *subtract-θ reset* (non-positive reset coefficient)	Exact reals; the CUDA audit uses the two-limb dyadic form	Neuronwise; no candidate-dependent recurrent input. No condition on $\beta(1+\theta)$ is stated, implied or needed
Thm. 7 (tight T-round bound)	Same LIF layer, constant currents $A_t = \theta(1+c)$	Exact reals	Lower bound for the stated iteration, not for every parallel LIF solver
Thm. 4 (class certificate)	Fixed linear class-valued readout over spike features	Sound enclosures; outward rounding suffices	Ties unresolved unless the declared tie rule is proved

Result	Model scope	Arithmetic	Output / execution assumptions
Thm. 5 (sparse exact repair)	Same layer, a known unresolved set U	Exact dyadic arithmetic (summary tree)	Repair cost charged separately from preprocessing
Thm. 6 (asynchronous windows)	Independent window solves with stale boundaries	Sound enclosures per window	Freshness conditions; a stale boundary is a counterexample (App. A)
Prop. 1 (lease invariance)	Homogeneous slots, transported parameters	Bitwise only if operand/reduction order is preserved	No slot-specific parameters or learned routing
Prop. 2 (reference preservation)	Finite DAG, fixed selected parent sets and deterministic combination functions	Inherits each node's local contract	Liveness needs stable source versions, feasible capacity, fair service
SHD certificate (§5.1)	One frozen quantized ridge readout, $T = 64$, 32 groups, count/8	Exact, FP64 and FP32 outward rounding	Zero wrong releases at every round; fallback for stalled coordinates
N-MNIST certificate (§5.1)	Same contract, 4-class prefix, 94.85% model	Same three arithmetics	Certification covers the released classes only, not dataset accuracy
Dyadic leak sweep (App. G.8)	Trained readouts at $\beta \in \{1/2,\ 3/4,\ 7/8,\ 15/16\}$	FP64 enclosures with directed rounding	Zero wrong releases at every β; release rate falls with leak
512-neuron interval study (App. I.3)	One frozen 512-neuron SHD layer, $\beta = 19/20$, $\theta = 1$, *reset to zero*	The reset-to-zero sound interval extension of Prop. 11 with Cor. 12; outward-rounded FP32/FP64	Thm. 4 over a sound box; *not* the Thm. 3 sandwich, which is stated for subtract-θ
Same-kernel control (App. H)	Single-model SHD request stream, 4–64-way concurrency, 24–96-step windows	FP32 fused kernel for both arms	One process per device; the architecture's cost is charged to the timed path
Repair width sweep (App. H)	W independent branches plus one fusion root, revision density 1	FP32 fused kernel; float execution reference	Repair and full re-execution bitwise identical; wall gains are host-side
The hull rule (App. I.2)	One frozen 512-neuron reset-to-zero recurrence, $\beta = 19/20$, $\theta = 1$, $T = 100$, official 2,264-sample test split	Exact one-step reachable box, checked in exact rationals and run in deployed FP32	Exact among sound rules that keep a box and take one step; all 2,264 released at median round 12; priced against our own sequential-sweep rule, not against another method

Result	Model scope	Arithmetic	Output / execution assumptions
Cost of the guarantee (Table 5, App. F.1)	Exact-dyadic contract models, and the one non-dyadic 512-neuron model outside it	Exact-trace proof, exact-dyadic scan, outward-rounded FP32/FP64 enclosure	Every certified row keeps the released class; ratios are against this paper's own plain forward on one device and boundary, with the plurality vote charged to every arm
Full-trace consumer and the exact fallback (App. G.3)	The same 512-neuron certificate, under the exact-spike contract instead of the class contract	Outward-rounded enclosure rounds with warm-started exact-dyadic solves	1,954 of 2,264 (86.31%) need one exact solve each at a median 1.41 ms; the class contract needs none at $K \geq 16$
Fallback frequency (Table 8)	All 2,264 official SHD test samples, both enclosures	Enclosure rounds to a round budget, then exact-dyadic solves	K here is the enclosure round budget, not the map-call budget of Table 10; needing a fallback and abstaining are different events
Memory ledger (Table 9)	The serving loop's own accounting on the RTX 4060 Laptop	Measured allocation rather than a modelled estimate	Capacity bounds execution slots, not memory; live tasks drive it, and the declared per-task figure underreports by $4.43\times$
Mixed class/full-trace consumers (Table 28)	One shared producer, two consumers, 128 requests, capacity 4, FCFS, $K = 8$ window	Certified scan bitwise-identical to the fused backend in spikes; membranes differ by at most 3.7×10^{-8}	A negative, workload-specific result: the release point is content-determined, so no earlier start is bought here; not a general statement about other readouts or loops
External runtimes at matched accuracy (App. I.9)	One frozen 512-neuron model, batch 2,264, input already device-resident	FP32, one shared weight tensor, CUDA events, paths interleaved	Accuracy matches exactly on every path; batch-1 dispatch-bound figures are deliberately not reported
Stream-level intervals (Table 27)	18 generated independent-task streams plus the anchor, one RTX 5090, FCFS, 24-step windows	FP32 fused kernel in every arm	The replication unit is the stream, not the repeat; host thread budget moves ratios by 1.2–1.5×, and no ratio is carried to another table

G.2 Scope conditions and costs behind the limitations

Table 7 above gives the assumptions each result needs. This subsection states, for each limitation the main text names, what follows from it and where the cost is measured.

The certificate is neuronwise and applies once a layer's input is fixed. It therefore does not cover recurrent candidate-dependent input, and it certifies a class rather than unchanged logits, so a tie stays unresolved unless the declared tie rule is separately proved. Its bounds are real-arithmetic:

validated arithmetic stays inside the dyadic conditions, and outside those conditions the certificate either withholds a class or falls back to an exact solve (Appendix G.3 measures how often the fallback is actually invoked, and under which budgets).

The repair and certificate results are work, depth and correctness statements, not latency bounds. Every cost attached to them is stated with its measurement: certified release costs $553\times$ the authors' own fastest no-guarantee arm's wall clock on the full test split, and no amount of downstream work closes that gap (Appendix G.9); selective repair repays its bookkeeping in proportion to how often inputs are revised, which at five batches and two capacities does not separate it from restarting the affected root (Appendix E.1); and capacity bounds execution slots rather than device memory, so task-private state remains allocated while a task waits (Appendix H). On an independent-task stream ELYSIA retains 0.48–$0.65\times$ ELYSIA-P over 18 streams, which is the workload our own ELYSIA-P exists for.

G.3 Fallback frequency, and what the state costs in memory

How often is the exact fallback used? The floating-point certificates in Appendix G.8 do not run the exact dyadic solver; they run outward-rounded enclosure rounds and fall back to the exact solve only for coordinates that are still unresolved when the round budget ends. Under the class-release contract the paper reports, the fallback is invoked for *0 of 2,264* samples at both FP32 and FP64, because every sample certifies within the 14 rounds the loop runs, and the same holds for any budget $K \geq 16$. A consumer that needs the exact spike trace rather than a class is a different contract: 86.31% of samples (1,954 of 2,264) are released before their coordinates are fully resolved, so that many exact solves would be needed, at a median 1.41 ms per sample warm-started (3.16 s FP64, 3.41 s FP32 for the set, against 17.66 s and 15.75 s for the enclosure rounds). Truncating the round budget forces fallback even under the class contract: 1.24% of samples at $K = 8$, 17.98% at $K = 4$, 57.33% at $K = 2$ and 95.14% at $K = 1$, with the exact solve contributing 0.2%, 3.6%, 10.7% and 16.5% of the certificate's wall time (Table 8). Here K is the *enclosure* round budget of the validated-arithmetic path, a different counter from the map-call budget K of Table 10: needing an exact fallback and abstaining are different events, so the two tables' K columns are not comparable. The hybrid verdict never disagrees with the exact reference. Run-to-run enclosure spread was $\pm 20\%$ and the FP32/FP64 timing order flipped between runs, so no precision-versus-speed claim is made.

Round budget K	samples needing exact fallback	share	exact-solve seconds FP64 / FP32	share of that precision's total
1	2,154	95.14%	3.487 / 3.769	16.5% / 19.3%
2	1,298	57.33%	2.109 / 2.284	10.7% / 12.7%
4	407	17.98%	0.672 / 0.713	3.7% / 4.3%
8	28	1.24%	0.0438 / 0.0503	0.2% / 0.3%
16	0	0%	0	0%
natural stop (14 rounds)	0	0%	0	0%

Table 8: Exact-dyadic fallback frequency and cost against the enclosure round budget, over all 2,264 official SHD test samples (identical counts for both enclosures). The exact-solve seconds are warm-started solves of the fallback subset only, reported separately for the FP64 and FP32 enclosures, and the share is of that precision's own enclosure-plus-fallback wall time. At the budget the certificate actually uses, no sample needs the fallback.

What does retained state cost, and does capacity bound memory? Per task, the logical record is a membrane [24] in FP32 plus cursor and generation counters: *112 bytes*. The real tensor footprint is *496 bytes* per task, because the physical pool adds owner and slot arrays; the declared figure therefore under-reports by $4.43\times$, and ELYSIA-P's declared and real figures agree at 112 bytes. The serving loop's own accounting inherits the same bias (it reports 7,168 bytes for 64 live tasks against a measured 9,216–31,744 bytes), so no memory estimate of the form live tasks $\times$ `per_task_state_bytes()` is used anywhere in this paper. Memory is bounded by live tasks, not by capacity: the logical state grows linearly with live tasks, an admission scorer of shape [tasks, pool] costs about 0.5 kB per live task per unit of capacity (2.0 MB at 1,024 tasks with capacity 4, 33.1 MB with capacity 64), and input tensors dominate at the full split (406 MB); total peak allocation rises from 11.1 MB at 4 live tasks to 822.1 MB at the 2,264-sample split. Capacity

fixes only the physical pool and the model-side tensors, which are independent of the number of live tasks. The claim that capacity bounds active execution slots rather than memory is therefore confirmed by measurement, and Table 9 gives the ledger.

Where the architecture's time goes. On the RTX 4060 Laptop GPU (24 SMs, shared desktop session, non-boost clocks) the same kernel measured on the device takes 7.43 μs at capacity 4, 7.82 μs at 16 and 9.21 μs at 64 (a trivial-kernel floor of 6.28 μs on this device), while the Python wrapper that launches it costs 431–513 μs per call, i.e. 47–69$\times$ the device time. Capacity also maps to only 1, 2 and 6 blocks of the device's 24 SMs (4.2%, 8.3% and 25.0%), although the occupancy API reports room for 6 blocks per SM; the GPU is therefore never the constraint at these sizes. This is the physical explanation of the architecture's cost measured by the same-kernel control: its overhead is host-side dispatch and bookkeeping around a sub-10 μs kernel, not GPU work. The larger device time quoted in Section 3 (14.6–14.8 μs) is the same quantity on a different device, the RTX PRO 6000 of Table 21; the two ladders come from different hardware and neither is used to compare executors.

Component	grows with	per unit	at 1,024 live tasks, capacity 64
Task logical state	live tasks	112 B	114.7 kB
Physical pool (owner, slot)	capacity	384 B per unit capacity	24.6 kB
Serialized checkpoint	live tasks	≈138 B per task ≈0.5 kB per task	141.7 kB
Admission scorer	live tasks × capacity	per capacity unit	33.1 MB
Published artifacts	admitted tasks per window	80 B per admitted task	5.1 kB
Model weights (parameters)	capacity / model width	85.7–270 kB	(capacity-set)
Input tensors	requests	per-sample window	406 MB (full split)

Table 9: Memory ledger of the architecture. The declared per-task figure (112 B) omits the physical pool, so the measured per-task footprint is 496 B; live tasks, not logical capacity, drive total memory.

G.4 Internal certificate and repair comparisons

Terminology. Several of our terms are borrowed, and we name the field each comes from rather than leaving the reader to infer it.

From validated numerics and interval analysis. An *enclosure* is an interval guaranteed to contain every value a quantity can take; *outward rounding* moves an endpoint away from the true value by one ulp so that the enclosure stays sound; a *box* is the same object in more than one coordinate; and the *hull* of a set is the tightest interval containing it, so our hull rule returns exactly that interval rather than a relaxation of it.

From verification and abstract interpretation. A bound or a rule is *sound* when it never excludes a reachable value, and we use *certificate* in the verification sense: the computation emits sound bounds together with its conclusion, and the final margin test is checkable from those bounds alone in integer arithmetic. The bounds themselves are re-derivable, and the checker below re-derives the enclosure rules with exact rationals rather than floating point. *Dyadic* is the number-theoretic term for a rational of the form $k/2^n$; the exact-dyadic conditions are the six arithmetic properties listed in Appendix G.8 under which every quantity in the trace is such a number and the trace is exactly representable.

From systems. We use *reference* for the result the full, unabridged computation would produce — the object every guarantee in this paper is stated against; *artifact* for what a subtask publishes; *contract*, in the artifact and consumer contract, for the promise a producer makes to a consumer, which is a different thing from the exact-dyadic conditions above; *lease*, from distributed systems, for the grant of whole neuron groups to one task; *lineage*, from data provenance, for the set of outputs whose computation read a given input; and *replay*, from record-and-replay debugging, for re-executing a recorded sequence.

A standard-library rational checker exhaustively evaluates 2,046 candidate vectors in the tight construction, verifies its T-round behavior for $T = 1, \ldots, 32$, and tests 250 rational traces with $T \leq 24$, varied leak, threshold, signed inputs and initial states. Across 3,587 rounds, sandwich, prefix,

arbitrary-partition window equality, readout bounds and sparse repaired spikes/final states all agree with the causal reference. Another 11,511 updates verify the enclosure invariant with exact and nonzero rational radii. These counts describe deterministic property checks, not independent trials of GPU reliability. The nonzero-radius test deliberately permits unresolved ties.

G.5 Learned readout detail and historical audit

SHD preprocessing, arithmetic and release audit. Channels are grouped by $\min(\lfloor 32c/700 \rfloor, 31)$, and events outside the fixed 1.28-second window are omitted (73 training events, zero test events). The four feature blocks average each channel's spikes over 16 consecutive bins. Ridge regression fits one-hot labels with an unpenalized intercept; weights and intercept are then rounded to multiples of $1/1024$. The quantized model has training accuracy $5569/8156 = 68.28\%$. No encoder parameter is trained. The exact reference represents every LIF state using arbitrary-precision integers with common denominator 2^{67}. Each input numerator is a multiple of 2^{64}; division by two remains exact through the 64 steps and is asserted. Readout scores have common denominator $16 \cdot 1024$. The pairwise margin test in Appendix A.3 uses signed 64-bit integer arithmetic. Before evaluation, arbitrary-precision checks bound every product, difference and partial sum: score magnitude is at most 219,344 and pairwise margin magnitude at most 333,360 for the fitted model. Both are below $2^{63} - 1$, so the readout calculation is also exact. The test requires a strictly positive lower margin against a lower-index competitor and a nonnegative margin against a higher-index competitor, matching the declared smallest-index tie rule. No uncharged extra map evaluation is used.

Before fitting, a rational recurrence written by the authors, not a third party, checks 8,192 time/channel values; a random integer readout is checked over 64 rounds, including an all-zero tied readout. An initial audit of the first 128 official test samples was extended to all 2,264 without changing model or hyperparameters. The full run reproduces every pilot sample's per-round record and model prediction. The package preserves both result files, the quantized weights, input hashes, package versions and every sample's per-round certificate/mismatch record. All rounds compare certified classes with the exact quantized model prediction; dataset accuracy and class invariance are separate outcomes.

K map calls	Class certified	Spikes certified	Abstain	Fixed-K wrong class
2	241	0	2023	403
4	1687	393	577	140
8	2233	2079	31	8
12	2264	2254	0	0

Table 10: Full SHD test split: counts of samples, with zero wrong certified classes at every round. Wrong class means disagreement with the exact causal *model prediction*, not the dataset label. Abstention withholds an uncertified class. All spike tensors certify by $K = 17$, the longest full-trace release wave, beyond the budgets tabulated here. K here counts certificate map calls and is not the enclosure round budget of Table 8.

Internal release-policy comparison: three learned-readout views. The 0/1/2-bin shifts are fixed before evaluation, with leading zero padding and trailing truncation. All three branches share the already fitted readout; they are not independently trained models. Each root requires all three committed class payloads, with plurality voting and smallest-index ties. No reference labels are passed into root execution. The full-spike policy also checks every committed full tensor against a causal solve the authors wrote. Four full-test repetitions alternate policy order AB/BA after warming both policies on four roots. All repeated artifact payloads and logical release traces are identical. The source records every root's parent classes, certification rounds, root output, and dataset label separately.

Class release reduces the three-view branch map calls of Table 11 by 35.70% and the shared-producer maps of Table 12 by 29.18%; both release 1,662 of 2,264 roots earlier, and the two percentages differ because the denominators differ. Its extra readout-bound work is included in the reported elapsed time. This small learned workflow demonstrates contract composition, with material CPU timing variation; it does not establish optimized hardware performance, capacity-pressure behavior

Metric	Class release	Full spike release
Total branch map calls	25,652	39,897
Mean root certification round	4.160	5.874
Wrong parent / root artifacts	0 / 0	0 / 0
CPU map time, median seconds	20.677	33.185
CPU certificate/readout, median seconds	1.030	0.269
CPU total time, median seconds	21.760	33.560
CPU total time range, seconds	15.954–25.956	27.384–37.896

Table 11: Internal release-policy ablation, exact-integer CPU prototype on the full SHD test split. Each view has a separate producer. Timing includes maps, certificate/readout, dispatch, artifact checks and root consumption, with input preparation and the authors' own reference outside the timed region. Component medians need not sum to the total median. Four repeats characterize this implementation only; these are not GPU latency results.

or a deep independently trained SNN pipeline. Its root accuracy is 54.99%, compared with 57.51% for the unshifted view.

Reference versus implementation costs. Theorems use exact maps and work-efficient affine scans. The historical windowed GPU implementation uses dense local matrices and an associative carry scan. The rational sparse-repair reference uses sequential Python loops to build segment summaries. Neither implementation establishes the ideal parallel work/depth bound as a measured hardware result. FP32/FP64 spike arrays are not bit-packed. Input storage, both iterates, boundary storage, scan temporaries and readout state must all be counted in a memory comparison.

G.6 Internal release-policy comparison with a shared producer

`experiments/shd_shared.py` keeps the original quantized model, all official test inputs, three consumer transforms, tie rule and root function. For the class policy, one base iterate advances while each uncommitted consumer tests the shifted sound box. The known prefix includes both the producer's certified prefix and the inserted zeros. Already committed classes are immutable; the producer retires only after all three commit. The full-spike policy completes that same producer, then computes the three readouts. The causal policy evaluates the base recurrence once, then uses those same transforms, readouts and root. No policy reads reference predictions as inputs.

Four repeats use class–full–causal, then the reverse order, twice. Warm-up covers four roots per policy. Data loading, event binning, model validation, reference construction, the authors' own validation and JSON writing are excluded. Producer calculation, view allocation, certificate/readout, vote, dispatch and reference safety assertions are included. These are exact-integer CPU implementation diagnostics, without CPU pinning or exclusive reservation; no confidence interval or inference-serving p95 is estimated from four repeats. The initial repeat is retained despite being slower.

Median producer, transform, certificate/readout and vote components are 2.479/0.053/0.420/0.002 s for class release, 3.472/0.005/0.037/0.002 s for full spikes and 0.583/0.004/0.030/0.002 s for causal execution. Component medians need not sum to the median total; the JSON also records dispatch/check residuals. Identical input/checkpoint hashes and every parent/root artifact are checked against the prior experiment. Sharing is a verified special case, not a general claim that dependent SNN nodes can be merged.

G.7 Calibrated confidence-threshold early-exit baseline (ELYSIA comparison)

The same frozen encoder/readout pipeline and the same streaming prefix features evaluate a confidence policy that releases a class when the prefix-based top-two score margin exceeds τ. The threshold is calibrated on the 8,156-example official training split: τ is selected for zero observed wrong releases at maximal early release and then frozen for the test split. The certificate of Theorem 4 is the only policy in the comparison that combines zero reference disagreements with substantial early release.

Shared-producer study (2,264 test samples)	Class release	Full spikes	Causal
Producer Φ calls	9,418	13,299	—
Mean certification round	4.160	5.874	—
Median certification round	4.0	5.0	—
Roots before base full-spike certificate	1,662/2,264	—	—
CPU seconds, repeat 1	7.659	9.247	1.325
CPU seconds, repeat 2	2.960	3.522	0.622
CPU seconds, repeat 3	2.974	3.547	0.619
CPU seconds, repeat 4	2.944	3.524	0.611

Table 12: Producer Φ calls, certification rounds and repeat timings for the shared-producer study (all 2,264 official test samples, exact-integer CPU implementation; one common shared producer for every policy). Producer calls and certification rounds are recorded for the two releasing policies; the causal policy evaluates the base recurrence once and has no certification round. All repeats are from `results/shd_shared.json`; these measurements replace an incomplete comparison, without discarding the original separate producer records. Class release cuts producer maps by 29.18% with zero parent/root class mismatches: the 6,792 parent classes and 2,264 root votes agree with the full-spike release. The single-view full-test study reports the corresponding 5.874-to-3.780 mean-map reduction in Appendix G.5.

Release policy	Early released	Wrong at release
Certificate (Thm. 4), same readout	86.31% (1,954/2,264)	0
Margin confidence, calibrated on train ($\tau = 8499$)	1.46% (33/2,264)	0
Margin confidence, uncalibrated operating points	up to 100%	95.1% of early releases

Table 13: Calibrated confidence-threshold baseline versus the certificate on the frozen pipeline (2,264 official test samples, exact integer arithmetic). τ is selected on the 8,156-example official training split for zero wrong releases at maximal early release, then frozen for the test split; it is on the integer-score scale. Early released = released before all spikes resolve. Wrong at release counts released classes that disagree with the full-horizon reference prediction, which the certificate guarantees to be zero by Theorem 4. The certificate row uses the same frozen readout as the calibrated margin policy (outward-rounded FP64 enclosures); under the exact dyadic scan of Appendix G.8 the same certificate releases 83.75%, and a 512-feature readout reaches 89.53% at zero wrong releases. The third row is the paper's uncalibrated margin policy, retained for reference. Learned adaptive stopping is instead compared in full in Figure 2 and Appendix G.9, on this same frozen pipeline: SEENN-I and SEENN-II, a learned halting head, the calibrated margin rule and SpikeCP are each swept over their own knob, and every one of them changes classes to exit early while the certificate does not. Those arms are re-implementations of the published methods rather than the authors' own code, which bounds what the comparison shows.

The calibration run, the frozen weights and the runner live in `evidence_external/experiments/results_confidence_baseline_20260914/`. Here and below, `evidence_external` names a directory of the evidence package that is external to this paper's main source tree — it is not a third-party provenance marker, and every campaign cited under it is the authors' own.

G.8 Validated FP32/FP64 certificate

The dyadic conditions of Appendix G is exact but restricted. This study replaces it with outward-rounded floating point: $\beta = 1/2$, $\theta = 1$ and the count/8 inputs are exactly representable, so the membrane recurrence is the only inexact operation and every one of its subtractions/additions rounds outward (`np.nextafter`), making the published bounds sound in real arithmetic. Readout arithmetic is exact in FP32 for the frozen model (weights on a 1024 grid, every intermediate below 2^{24}, verified). Table 14 reports the result.

The SHD enclosure argument (directed rounding, monotonicity, readout exactness bound) and runner live in `evidence_external/experiments/`

Precision	Early released	Wrong at release	Never certified	Median round
SHD, 2,264 test samples (57.51% model)				
Exact dyadic (paper study)	83.75%	0	0	3.0
Exact (accumulated bounds)	83.30%	0	0	3.0
FP64 (outward rounding)	86.31%	0	0	3.0
FP32 (outward rounding)	86.31%	0	0	3.0
N-MNIST, 4,157 test samples, classes 0–3 (94.85% model)				
Exact dyadic	98.82%	0	0	4.0
FP64 (outward rounding)	98.82%	0	0	4.0
FP32 (outward rounding)	97.93%	0	0	4.0

Table 14: Validated-arithmetic certificate on two datasets. Outward-rounded FP32/FP64 enclosures preserve the zero-error guarantee: every released class equals the exact reference prediction; on SHD FP32 and FP64 release exactly the same samples as each other (86.31%), 3.00 points more than the exact accumulated-bounds scan and 2.56 more than the exact-dyadic study, while on N-MNIST FP64 matches the exact rate (98.82%) and FP32 loses 0.89 points of early release (97.93%) without a single wrong release. Model accuracies are 57.51% (SHD) and 94.85% (N-MNIST, 4-class prefix). Stalled coordinates revert to the exact dyadic fallback (Appendix G).

Leak β	Early released	Wrong at release	Never certified	Median round	Reference accuracy
SHD, 2,264 test samples, one fixed contract					
$1/2$	86.31%	0	0	3.0	57.51%
$3/4$	81.36%	0	0	5.0	58.17%
$7/8$	71.51%	0	0	6.0	60.20%
$15/16$	60.47%	0	0	7.0	59.58%

Table 15: The leak sweep behind this paper's $\beta = 0.9375$ statement in Appendix F.1. The FP64 enclosure certificate is run at four leak factors with the rest of the contract ($T = 64$, 32 groups, count/8, $\theta = 1$) held fixed; every β releases all 2,264 samples, never abstains and records zero wrong releases, and the early-release fraction falls monotonically as the leak widens. The sweep fits the same frozen readout as Table 14 on the same 8,156-example training split and evaluates it on the same 2,264 test samples with the same enclosure path, so its $\beta = 1/2$ row reproduces that table's 86.31% enclosure result; only the leak factor changes across rows.

`results_fp32_certificate_20260914/`. This study fits its readout on all 8,156 training examples, so its test accuracy (57.51%) differs slightly from the compound graph's coarse parent of Appendix E, which fits on 7,340 and reserves 816 (56.71%); the two are separate frozen artifacts under the same conditions.

The N-MNIST study reuses the identical dyadic conditions (T=64, 16 ms-equivalent bins, 32 channel groups, count/8, $\beta = 1/2$, $\theta = 1$) and the identical ridge/quantization pipeline; its training set is a class-balanced prefix of the official training archive (600 samples per class, classes 0–3, fixed before any fitting), and certification covers the first 4,157 test samples of those classes. Those 4,157 are exactly the samples the decoder yields with labels 0–3 from the downloaded official test archive, 157 more than a balanced four-class prefix of 1,000 per class would give; we report the decoded prefix as run and do not re-weight it, so this study is a prefix of the decoded archive rather than a balanced subset, and the SHD study of Table 14 remains the primary certification evidence for the guarantee. Evidence: `evidence_external/experiments/results_nmnist_certificate_20260914/`. General β and non-dyadic inputs remain the roadmap step toward fully general validated arithmetic.

The certificate on a trained spiking layer. Every certificate study above uses a single-layer model with an untrained encoder, which leaves open whether the machinery survives a trained layer inside a trained graph. We ran it on task F of the learned $A + B \rightarrow F \rightarrow H$ graph of Appendix E.1, whose frozen weights, inputs and reference predictions are hashed in `implementation_multilevel_learned/`. Task F is the right target because its input is the pair of score vectors that A and B publish on commit: once both parents commit, F's input trace

is fixed and candidate-independent, which is exactly the hypothesis the class certificate needs, so the existing theorem applies without new mathematics.

We first reproduced the frozen reference on all 2,264 roots ($1{,}530/2{,}264 = 67.58\%$, agreeing with the deployed GPU run on every root, maximum score difference 4.8×10^{-6}), then ran the outward-rounded FP32/FP64 enclosure on F's 32-neuron subtract-reset LIF ($\beta = 0.75$, $\theta = 1$, 24 steps) with the trained readout H. Every root releases a class, none is never-certified, and *zero* released classes disagree with the exact reference. An audit run by the authors finds no violation of $\underline{z} \le z^* \le \overline{z}$ in any round under either precision, and FP32 and FP64 agree on the certification round for 2,263 of 2,264 roots. 2,245 roots release before all coordinates resolve (99.16%); under the shipped convention, which excludes roots whose trace never fully resolves inside the budget, 1,207 do (53.31%), at a median certification round of 5 and a maximum of 23.

The scope caveat is the arithmetic. Because F's currents come from a $\tanh$ they are real-valued, so the exact-dyadic conditions does not apply: the certified object is the real-arithmetic trajectory on the exact FP32 currents and weights, not the FP32 instruction stream. Here the two agree exactly, with zero mismatches over 1,738,752 coordinates between float64 and the deployed float32, but that agreement is an empirical check rather than part of the certificate, and a general non-dyadic conditions remains the roadmap step below.

Model scale on SHD. Every SHD accuracy figure elsewhere in this paper comes from a model that meets those conditions (32 neurons in 32 channel groups), a scale chosen so that the group-lease and certificate machinery is cheap to exercise; that leaves open whether the reported accuracies describe the task or describe the model. We therefore trained a separate family of 512-neuron models on the identical frozen pipeline: the same official `shd_train.h5` and `shd_test.h5` archives (hashes recorded alongside the artifacts), the same per-sample time binning, and the same 2,264-sample test split. Each arm is $\mathrm{Linear}(700, 512) \to \mathrm{LIF}(\tau) \to \mathrm{Linear}(512, 20)$ applied to the pooled hidden spikes, with 368,640 parameters, Adam under a cosine schedule, and seed 0, for 120 epochs (80 for two arms). Each arm reports its best test epoch, which is mild test-set selection and is what the trainer does. Table 16 lists the arms; the best uses $T = 100$ and $\tau = 20$, at $1{,}742/2{,}264 = 76.94\%$ with a 5.5% hidden spike rate.

Arm	T	τ	Decay input	Best test accuracy
T3	100	20	no	**76.94%** (1,742/2,264)
M1	100	2	no	74.34%
T4	100	10	yes	74.25%
T2	100	10	no	73.72%
M2	250	2	no	73.67%
S3	100	2	no	73.59%
T1	100	5	no	72.61%
S2	100	2	no	67.80%

Table 16: A capacity probe on the same SHD task and the same frozen test split, at 512 hidden neurons instead of 32. All eight arms share one spiking layer and 368,640 parameters. They differ in the membrane time constant τ, in whether the input is decayed, and in readout pooling (sum for every arm except S2, which uses the mean); M1 and M2 additionally use input dropout 0.05 and a wider time-shift augmentation. These rows are not comparable to the 67.58% compound graph of Appendix E.1 or to the 57.51% certified model of Table 14: the architectures and the fitting protocols differ, so the table probes model capacity and nothing else.

The probe bounds the claim in both directions, and we state both. It shows that the 57–68% figures above are a property of the model that meets those conditions rather than of the execution system: a $16\times$ wider layer moves the same task by about nine points while the serving path, the lease protocol and the artifact contracts are untouched. It equally shows what this evidence cannot support. The family is feed-forward with one spiking layer and a linear readout on pooled spikes, so it is not comparable to the recurrent and deeper architectures that reach the 90–96% range on SHD, and we do not present 76.94% as a competitive entry in that literature. The family also carries no certificate: the enclosure of Appendix G.8 is instantiated on the 32-neuron trained layer of task F, and no certified path exists for these arms. The probe answers the capacity question and leaves the accuracy question to work that trains deep SNNs.

Provenance: the 76.94% checkpoint was re-executed after the training instance was retired, on different hardware and a different torch build (`2.11.0+cu128` against the original `2.8.0+cu128`), and against a from-scratch reimplementation of the LIF recurrence rather than the training framework; it reproduces $1{,}742/2{,}264$ exactly.

G.9 The certificate against early-release methods, across depth and models

This section carries the evidence behind Figure 2 and §5.1, from four separate experiments, as one set of results rather than four studies. Everything here is measured; the two propositions are exact and are proved for any model.

Why the release decision is not a scan: four obstructions. The release decision is a nonlinear fixed-point iteration, not a scan of the trace, and the difference is not cosmetic — it is what makes the worst case $\Theta(T)$ rounds, attained by the paper's own hard construction. Four independent results, all machine-checked in exact rational arithmetic (`experiments/margin_audit_20260917/`).

(i) The associative monoid of Thm. 5 presupposes U. The segmented-scan summary is associative and reproduces the trace exactly when it is given the true U_K ($151/151$ trials), and it is wrong in $49/49$ trials that carry a genuinely wrong coordinate once a single wrong coordinate is dropped from U. Across those trials U held 325 coordinates that were already right and 54 that were genuinely wrong, so a U that is merely *plausible* is not enough. The scan consumes U; it cannot produce it.

(ii) U_K is not a function of the evaluated prefix. Two instances with identical currents on $[0, K)$ and identical exact prefixes, differing only on the tail, give $U_K = \varnothing$ against $U_K = \{3\}$ at $K = 2$ and opposite decisions — release against abstain — on the same readout. So no rule that looks only at the prefix can decide release, however long the prefix.

(iii) The worst case is the paper's own family. On the tight construction of Appendix A.2 the family gives $U_K = [K, T)$ and pinned set $[0, K)$, verified exhaustively for $T \le 12$: the first exact iterate is at $K = T$, so no budget below T resolves the trace and the release decision needs $\Theta(T)$ rounds.

(iv) With a last-coordinate readout the margin test never releases early. On the same family, a readout reading the last coordinate abstains for every $K < T$; it releases exactly at $K = T$ when the reference last spike is 1, and at $T = 4$ and $T = 6$ it abstains even at $K = T$ because the margin is an exact tie.

Together these say the release decision cannot be reduced to a scan or to a prefix rule, and that its depth bound is the iteration's own. That bound is a statement about the *ideal parallel route* to the release decision, not about the rule this paper ships: our sequential-sweep rule is sequential, so its rounds are wall-clock rounds rather than depth, and the hull rule changes the constant, not the shape. Quantitatively, the ideal parallel depth $K\lceil\log_2 T\rceil + \lceil\log_2 T\rceil$ exceeds the serial trace depth at the scales this paper measures — $75 > 32$ at $(T, K) = (32, 14)$, $90 > 64$ at $(64, 14)$, $108 > 64$ at $(64, 17)$, $126 > 100$ at $(100, 17)$, $259 > 100$ at $(100, 36)$ — and wins only from $(256, 17)$ on ($144 < 256$). The $K = 36$ cell is the hull rule's own round count from Table 1, so the more rounds a rule needs, the worse the ideal parallel route does: improving the release set makes this particular bound worse, not better. Checked: 0 scan-round violations over the round sequences examined, 0 associativity violations over 151 trials.

"Early release" is three different claims, and only one is saved work. The certification round is the quantity a reader will be tempted to read as a latency win, so we measured the three senses separately on the same certificate and the same $2{,}264$ roots, and they do not agree.

(A) Exact tail. Round k pins the first k coordinates to the exactly resolved reference and bounds the remaining $T - k$ with the sound causal enclosure over the known currents. On the $T = 256$, 512-neuron retrained model this pins every coordinate in one pass — 0 unresolved of $296{,}747{,}008$ coordinate-round checks — so all $2{,}264$ classes certify at round 1 and $2{,}264$ of them are "early" against a 256-round budget by this reading. That number is real and it is also the reading under which the certification round is *not* a work saving: the enclosure pass that pins the trace costs a full sweep over the horizon.

(B) Unknown tail, the deployer's question: round k pins the first k coordinates and leaves every later coordinate in $\{0, 1\}$, so the class must be decided without evaluating the rest of the stream. This

is strictly more conservative than (A). On the shipped readout under the sequential-sweep rule this paper ships it certifies nothing before the last step: 0 of 2,264 roots release early, and all 2,264 certify only at $k = T$; that same sequential-sweep rule closes 499 of 2,264 proofs within its 60 rounds in the exact-tail sense of (A) (Table 1). That measurement is a fact about one readout and one rule, and not about the mechanism, because the same deployed recurrence releases all 2,264 classes at a median round of 12 under rule the hull rule of Appendix I.2. Deciding a class before the stream ends is therefore available; what it needs is a release rule that reads the reachable membrane range, which the hull rule does and the shipped interval widening does not.

(C) Parallel rounds. Iterating the sound interval extension of Proposition 11 in parallel, the Theorem-3 *agreement* channel contributes exactly the advancing prefix and nothing beyond it. At budget K the prefix $t < K$ is exact, so that channel pins 1,159,168 coordinates per round — 1,159,168 ($K = 5$), 2,315,121 ($K = 10$), 4,624,518 ($K = 20$) and 9,225,191 ($K = 40$), each inside a prefix of exactly that size and nothing outside it. This is *not* the Jacobi of Thm. 3: that theorem is stated for the subtract-θ reset, and its sandwich does not transfer to reset-to-zero. The rule measured here is the sound interval extension of Prop. 11 iterated in parallel, with Cor. 12 supplying what replaces order reversal; only strict lower triangularity, and with it prefix exactness, is inherited from the theorem's structure.

The *enclosure*, by contrast, pins far outside the prefix, and it does so from the first round. Re-measured on the deployed 512-neuron recurrence at $\beta = 19/20$, $\theta = 1$, $T = 100$ and reset-to-zero, over $2{,}264 \times 100 \times 512 = 115{,}916{,}800$ coordinates, our sequential-sweep rule pins 91,432,767 coordinates at round 1: 1,159,168 inside the prefix and 90,273,599 outside it, with 0 wrong pins and 0 lower or upper enclosure violations. By round 60 it has pinned 114,172,069 of them, 44,621,989 outside the prefix, at 0 violations. A reader who takes the prefix clause as a statement about the mechanism will therefore get the mechanism wrong: the two effects are different objects, and only the agreement channel is confined to the prefix.

The published certification round of the 512-neuron model in Figure 2 is sense (A): it indexes the round at which the margin test passes given the exact reference prefix, and the shipped certified path computes that exact-reference serial LIF for the whole batch before any round runs, in both its full and its early-release modes, because the prefix is pinned to it. So the certified arm executes the full trace and then the enclosure rounds and the exact-integer readout on top of it, which is why it costs 553× the authors' own fastest no-guarantee arm rather than less, and why Figure 2 draws it at the trace's own work with the certification round carried in the annotation instead of on the axis. On a reset-to-zero neuron this certificate gives proof rather than parallelism: it proves the answer already exists within a budget that gates on the trace, so it pays the trace and is costed above the authors' own cheapest no-guarantee path rather than below it.

One pipeline, every method. Five method families and two of our own controls ran on one frozen encoder/readout pipeline: $700 \to 512$ LIF ($\tau = 20$, reset to zero, $\theta = 1$) $\to 20$ over the summed hidden spikes, $T = 100$ timesteps, the official SHD test split, all 2,264 samples, one checkpoint and one input cache shared by every arm. The model is a 512-neuron checkpoint at $1{,}742/2{,}264 = 0.7694$. Three correctness gates ran first: the plain forward reproduces the checkpoint's recorded test accuracy to the last digit; an independent float64 reference agrees on $2{,}264/2{,}264$ classes and on all 115,916,800 spike coordinates; and every arm consumed the same weights, inputs and sample count, re-asserted inside each arm's own row. Each knob was swept over its full range — SEENN-I over 21 values of α (its released-code rule, no halting network), the halting head over 19 values of τ, the margin rule over 9 values of δ, SpikeCP over four p_{targ} and both stopping rules, fixed truncation over all 100 values of T' — and the best value of each is the one reported. The margin rule's threshold is calibrated on the held-out split and its target error rate is set there, so this is a favourable instance of that family rather than its default. Numbers are class changes against the full trace on the same 2,264 samples.

Why the zero-wrong column cannot be earned by the model. The certificate withholds a class unless the candidate beats every competitor by more than the uncertainty on the unresolved coordinates (Thm. 4). That test never reads the label, the weights or the data distribution, so its product is forced:

Proposition 8 (The zero-wrong property is created by the margin test)**.** *Fix any datum d, any budget $K \geq 1$, any readout (b, M) and any index set of classes. If the margin test of Theorem 4 releases*

Arm (best setting)	Changed classes	Mean timesteps	Note
Full trace, $T = 100$	0	100.00	the reference answer
Fixed truncation, $T' = 90$	91	90.00	no guarantee; 150 at the holdout-selected $T' = 82$
SEENN-I, $\alpha = 0.99999$	29	78.12	1,349 at the paper's default $\alpha = 0.9$
SEENN-II (REINFORCE policy)	919	37.14	the paper's trained policy
Learned halting head, $\tau = 0.9999$	94	85.83	our control, not from a paper
Margin rule, $\delta = 10^{-4}$	5	86.77	calibrated; 84 at $\delta = 0.05$
SpikeCP, $p_{\text{targ}} = 0.99$	159	72.12	285 at $p_{\text{targ}} = 0.90$
Per-sample oracle exit	0	29.92	not deployable; the ceiling for any early exit
Certificate (Thm. 4)	**0**	**12.14**	2,264/2,264 released by the reset-to-zero enclosure, median round 11, max 36

Table 17: What early release costs in answers on the frozen 512-neuron SHD pipeline (2,264 official test samples, one checkpoint and one input cache for every row). "Changed classes" counts test classes that differ from the full trace's answer, which is the row the certificate is guaranteed to reproduce; it is not the model's error against the SHD label, which is unchanged at 522 for every row that reproduces the full trace. No swept setting of any baseline reaches the certificate's coverage at zero changed classes: the calibrated margin rule does reach zero, but only by releasing 1.46% of the split (Table 13), and at any coverage comparable to the certificate's every baseline here changes classes. The certificate's mean is a mean over *certification rounds*, which index when its proof closes and not how much work it skipped; the certified path executes the full trace, and Figure 2 therefore plots it at the trace's own horizontal position. The three senses of "early" and the wall-clock cost are separated below.

the class a on d at budget K, then $a = a^\star(d)$, the reference class, exactly. *Hence a released class cannot be wrong, as a pointwise statement, for every distribution on the data — whether or not $a^\star(d)$ equals the label, and whether or not the model is accurate.*

Proof. Soundness gives $|r^\star_a - \widehat{r}_a| \le e_a$ with $e_a = \sum_{t \in U_K} |M_{a,t}|$, so for the released a and every $j \neq a$, $r^\star_a - r^\star_j \ge \widehat{r}_a - \widehat{r}_j - e_a - e_j > 0$. Thus a is the unique maximiser. The inequality is strict by the test's own condition, so a tie is exactly the case in which the test abstains. □

The confirmation is a measurement on a model that cannot be good: with the label defined as the complement of the reference class, model accuracy is 0.00 by construction, and the same exact-integer certificate produced 46 early releases out of 200 samples with 0 wrong against the reference and 0 of the 46 released classes correct as labels. So every accuracy figure attached to a certificate row is the model's, not the certificate's: the certificate adds no correctness, it adds knowledge that this run is the full-horizon run. The release *fraction*, by contrast, is a property of the input population and moves by 70 points on one fixed model when only the input margins change, so a release rate is always reported with its population.

Abstention is a proof obligation, not caution.

Proposition 9 (The certificate's release set is the coarsest sound one)**.** *Fix K. Write $P_K(d) = \{t : s^{K-1}_t(d) = s^K_t(d)\}$ for the coordinates whose two sandwich bounds agree — the set the certificate releases at budget K, the complement of C_K in Eq. (3). Call a procedure R budget-K pointwise sound if for every datum d, every $t \in R(d)$ satisfies $s^\star_t(d) = s^K_t(d)$. Then for every such R and every d, $R(d) \subseteq P_K(d)$, and P_K is itself budget-K pointwise sound. So the certificate emits the coarsest sound release set at budget K, uniformly over all such procedures.*

Proof. P_K is sound by Theorem 3. For maximality, let $t \notin P_K(d)$ and let $R(d) \ni t$. Replacing d by a datum d' that agrees with d on everything the budget-K computation distinguishes, while having $s^\star_t(d') = s^{K-1}_t(d) \neq s^K_t(d)$, makes R release t on d and d' with two different reference values, so R is wrong on one of them. Such a d' exists whenever $U_K \neq \varnothing$, which is exactly what U_K records. An enumeration over 400 random systems found 70 instances with $U_K \neq \varnothing$ and exhibited the two-world witness for each. □

Proposition 9 says there is no slack to recover by a better test: releasing more requires giving up pointwise soundness, which is what a statistical rule does. The construction that makes this tight — $0 < \beta < \frac{1}{2}$, $\beta^2/(1-\beta) < c < \beta$, currents $A_t = \theta(1+c)$ — drives the released set at budget K down to exactly K coordinates of T, one per round, verified in exact rational arithmetic for $T = 5, \ldots, 12$; the same construction is where a statistical rule overtakes the certificate, and we report that below.

The guarantee composes across depth. A two-spiking-layer chain ($512, 512$ neurons, $630{,}784$ parameters, trained on the official train split only, $1{,}783/2{,}264 = 0.7875$ in float64, the same $2{,}264$ test samples) extends the certificate with no new hypothesis beyond finite precision: the hand-off and the affine step are accumulated at higher precision and rounded outward once. That condition is not optional — accumulating the hand-off in the working precision instead produced $99{,}860$ enclosure violations in $100{,}000$ trials with a worst error of 19.76, a re-association effect and not a last-bit one, while fp64 accumulation with one outward rounding produced 0. Composed this way the certificate releases $2{,}201/2{,}264 = 97.22\%$ at 0 wrong against both the exact fp64 reference and the deployed fp32 argmax, with 0 enclosure violations in fp64, and a three-layer chain matches it exactly; the single-layer arm on the same code releases $2{,}179/2{,}264 = 96.25\%$ at 0 wrong. Every released class equals the full trace's, so accuracy at release is the model's own. The composition *does* need one thing the single-layer statement does not spell out: the naive alternative of propagating an interval extension over the admissible parent set is sound — 0 violations — and does not contract; it reaches a fixed point after round 1 with $1{,}386{,}333$ of $6{,}553{,}600$ layer-1 coordinates unresolved and releases 0 of $2{,}264$ classes. Two concrete candidate traces per layer do contract, because antitonicity is a statement about corner evaluations.

A stronger model certifies more, not better. The certificate's guarantee does not depend on model quality (Prop. 8), but its coverage does. On a retrained 512-neuron model of the same pipeline whose leak again leaves the dyadic conditions, test accuracy is 0.8454 ($1{,}914/2{,}264$), against 0.7694 for the checkpoint above; the same certificate releases $2{,}264/2{,}264$ classes at median round 1 of a 256-round budget, 0 wrong, 0 wrong against the deployed argmax, with 0 enclosure violations in the corrected float64 extension; the literal float64 enclosure, uncorrected, shows 16 coordinate-rounds — *one* coordinate of $296{,}747{,}008$ held across 16 rounds — against 0 for the fp32 chain the model deploys. The released-class accuracy is the model's 0.8454: the certificate added no correctness. This is the strongest model in the family; the arm is the best of the configurations trained for it, chosen on the $1{,}000$-sample validation split held out of the official train split, with the test split used for reporting only.

What would have to be true for a statistical rule to win. We looked for a case where a calibrated or conformal rule beats the certificate, and found one, so we report it. On a population that concentrates half its mass on the tight construction above and half on a datum whose class is already certified at round 1, with a readout whose only weight sits on the last coordinate, the certificate cannot release the hard datum at any budget below T — its uncertainty on that coordinate is exactly the margin, so the test is never strict — and its release probability is at most $\frac{1}{2}$, while a rule calibrated on that population has risk 0 and releases both points at budget 1. The certificate is strictly dominated there, and no dominance theorem in either direction exists. Three things bound what this shows. The witness readout is a single coordinate; a trained readout with a large weight on the last timestep realises it, but we did not train one, so this is a construction and not a measurement. The winning rule's guarantee is valid exactly over that population and evaporates under covariate shift, whereas the certificate's is unchanged by shift. And with the certificate's own cost model its budget-K run is far more expensive than any K-step halting policy, which makes the counterexample stronger, not weaker.

What the guarantee costs. Measured in one exclusive device block with no peer compute process: the certificate costs $4{,}224.4$ ms for the whole test split against 9.4 ms for the full trace, 223.8 ms against 4.8 ms for a single request, and $553\times$ the authors' own fastest no-guarantee arm at batch $2{,}264$, taken against an arm at a different accuracy (0.751325, against the certificate's 0.769435). Against a downstream consumer the gap does not close, because the consumer's cost is added identically to every arm and the certificate has already spent its time: at a measured $1{,}464.6$ ms of downstream work the certified arm is still 4.2 s behind on a 9.3 ms forward pass, so the crossover is at infinity. It is a wall-clock win in exactly one regime, and we measured that too: against the same

certificate run to its round budget rather than releasing each root as it is proved, early release saves $1.62\times$ (4,229.1 ms to 2,606.9 ms) with identical released classes and identical certification rounds on all 2,264 roots. What the certificate gives on this recurrence is proof, and on a reset-to-zero neuron that proof is bought in proof obligations rather than in executed work.

What the guarantee costs against an external method. Every ratio above is a cost inside this paper's own stack: certified against this paper's own full trace, own fastest no-guarantee arm, own fused trace and own dense $T = 100$ forward, so $553\times$, $46.7\times$ and $362\times$ cost the certificate against cheaper paths we wrote; and the one external comparison in the same stack, at matched accuracy, reads 173–$226\times$ batched and 10.5–$15\times$ per request in the other direction. Both readings are costs of the guarantee, and the external one is the figure that says something about the field. Both arms of it ran at the same accuracy, 0.769435 ($1{,}742/2{,}264$), and all figures below are device-time means over the repeats of one locked session. Certified: $4{,}224.405$ ms for the whole $2{,}264$-sample split and 223.754 ms for one request. The SEENN-I re-implementation at $\alpha = 1.01$: 18.658 ms dense and 24.459 ms masked over the whole split, 14.812 ms dense and 21.386 ms masked for one request. The certificate is therefore 173–$226\times$ slower batched and 10.5–$15.1\times$ slower per request — which does not mean the guarantee loses by those factors; those factors are its cost. At the setting that matches the accuracy, SEENN-I never exits: 0 of $2{,}264$ samples stop before $T = 100$, mean executed timesteps 100.00, changed classes 0. Every setting at which it does exit changes classes — $2{,}040$ of $2{,}264$ at $\alpha = 0.3$, falling monotonically to 29 at $\alpha = 0.99999$, where its accuracy is 0.758392 — and the certificate changes none. So the external comparison reads: at matched accuracy the strongest external early-exit method measured here is 10.5–$15\times$ faster per request and 173–$226\times$ faster batched, and it reaches that accuracy only where it never exits; the certificate changes no class, so 10.5–$15\times$ is what it pays for a released class that always equals the reference. Two further external timings float around this literature and are *not* at matched accuracy, so they are quoted with their accuracy and are used as ratios nowhere in this paper: SpikeCP's singleton rule, 9.4 ms batched, is at 0.721290, and SEENN-I at $\alpha = 0.99$, 5.8 ms per request, is at 0.578622. The internal costs keep their own accuracies too: $553\times$ is against the authors' own fast arm at 0.751325 while the certificate is at 0.769435; $46.7\times$ is against the authors' own dense $T = 100$ forward at the same 0.769435; and $362\times$ is against the authors' own fused trace on the 57.51% model of Table 14, a different pipeline whose two arms are matched to each other there.

G.10 INTERNAL ABLATION: NUMERICAL-SOLVER COMPARISON, EXACT CUDA

Policy	Views	Active batch	Median ms	Min–max range ms
Fused LIF	1	32	10.440	8.931–12.253
Fused LIF	1	2,264	0.442	0.366–0.680
Fused LIF	3	32	10.220	6.795–12.405
Fused LIF	3	2,264	0.667	0.646–1.373
Full-spike scan	1	32	33.043	29.048–35.117
Full-spike scan	1	2,264	2.590	2.387–3.890
Full-spike scan	3	32	30.817	26.822–34.769
Full-spike scan	3	2,264	2.694	2.615–3.723
Class scan	1	32	30.563	25.810–35.028
Class scan	1	2,264	2.131	2.000–2.828
Class scan	3	32	28.302	25.577–37.345
Class scan	3	2,264	2.669	2.590–3.542

Table 18: Internal ablation: numerical policies, RTX 4060 Laptop: elapsed time to process all 2,264 test samples for the same three numerical policies, with 20 measurements per policy. The median is reported next to the min–max range of those same 20 measurements (one table, both statistics). Three-view policies all share one producer. Timings include numerical work, certificates/readouts, initialization, host dispatch/synchronization, class download and CPU voting. Persistent allocation/input upload is separately reported. These are total batch-workload times, not per-request latency quantiles; no request-tail or confidence interval estimate is inferred from these repeated batch timings. Raw ordered observations and source/input hashes are in `results/gpu_validated_backend.json`.

The CUDA backend fixes $T = 64$, 32 neurons, $\beta = 1/2$, $\theta = 1$, zero initial state and nonnegative int32 event counts divided by eight. An unsigned low limb and signed high limb represent a signed two's-complement numerator at denominator 2^{67}. A current count c has numerator $c2^{64}$; subtracting a candidate spike subtracts $8 \cdot 2^{64}$. Every term begins with 64 zero low bits. A length-64 candidate recurrence requires at most 63 decay shifts, so the shifted contributions remain integers. More specifically, before a scan stage with gap $g \in \{1, 2, 4, 8, 16, 32\}$, each partial sum contains at most g consecutive terms and is divisible by 2^{65-g}. Shifting it by g therefore remains exact because $65 - 2g \geq 1$; adding the two partial sums establishes the invariant for the next doubled gap. The causal recurrence's initial shift acts on zero; its last nonzero contribution also has at most 63 shifts. All intermediate sums are exact. A prefix's scaled magnitude is less than $2(\max c + 8)2^{64}$, safely within the signed two-limb range; the host checks input domains before launch. The high-limb bound is 292 on the test split. Since the low limb is unsigned, testing the high limb against eight exactly implements the numerator threshold $8 \cdot 2^{64}$, including equality. The readout reuses the separately verified int64 overflow bounds in Appendix G.5. These representability arguments concern the specified finite horizon and grid, and do not extend to arbitrary real leak or inputs.

Each scan block handles four neurons and all 64 time positions. Shared memory and barriers implement six Hillis–Steele doubling stages; each stage reads the prior published stage before writing the next. Thresholding the resulting candidate membrane produces a complete Φ iterate. A separate certificate kernel constructs per-view temporal counts, predicts candidate classes and tests all pairwise margins. Per-sample producer activity ends only after all declared classes commit, or after the complete spike certificate under the full-spike policy. A host completion counter synchronizes each global wave. This is a synchronous temporal GPU scan, not the asynchronous summary-tree implementation. Full-spike audit downloads select each sample's correct ping-pong buffer using its recorded retirement wave.

The exported DLL assumes calls arrive through the Python wrapper, which checks the nonnegative int32 current domain and readout dimensions before allocation. The C ABI is not an untrusted-input boundary; a caller that bypasses this wrapper must enforce the same checks to retain the two-limb overflow guarantee.

The fused LIF control runs one CUDA thread per neuron, uses the same two-limb arithmetic in registers, writes the same spike tensor and invokes the same readout/voting function. Every policy shares delayed producers. Full-test checks compare each causal and full-spike output tensor to the

Python-integer reference, and every class/root to the corresponding reference artifact. The exact class policy may return an approximate candidate spike tensor internally; it publishes only certified classes.

An adversarial audit compares 294,984 gate/class values across zero, constant, impulse and random currents and 18 arbitrary candidates. Under $q = 0$, constant current $1/2$ yields a time-63 voltage $1 - 2^{-64}$; initial current $9/8$ followed by $1/2$ yields $1 + 2^{-66}$. Exact threshold ties with $I_0 = 1$ and all-zero readout ties also pass. No floating tolerance or external replay fallback is used. This closes the implementation obligation for this exact dyadic backend; non-dyadic validated arithmetic remains a separate experiment.

The complete official test split is processed using active batches of 32 or 2,264, and either one or three consumers. All batch inputs are uploaded and allocated before timing; the active-batch parameter limits simultaneous computation, not total allocated input memory. Batches run sequentially. Correctness-audit passes precede measured runs. Ten cycles of causal–full–class–class–full–causal provide 20 observations per policy. Wall time includes Python dispatch, fresh task initialization, every numerical and readout kernel, per-wave completion synchronization, final class and release-metadata download, concatenation and root voting. File loading, event binning, compilation, reference construction and full-spike audit downloads are excluded. Allocation/upload costs are 33.192/17.786 ms for one-view batches 32/2264, and 31.691/17.954 ms for three-view batches 32/2264; these common setup costs are not charged to the steady-state table. CUDA-event spans are also recorded, but include host-induced idle gaps and must not be called summed active-kernel time. Peak allocated memory, request latency distributions, dependency-aware concurrency, selected-window GPU updates and serialized eviction are outside this benchmark's recorded paths.

A preceding four-observation development run is preserved separately (batch-2,264 class/full medians 3.076/2.939 ms versus 2.669/2.694 ms in the longer final run). The two runs' class/full order differs; all exact outputs and map counts agree. CUDA Compute Sanitizer reports zero memory errors and race hazards in the recorded checks (Appendix G).

G.11 Internal ablation: contract, window-update and release controls

We execute a 13-node graph with nine leaves, three intermediate nodes and one root. Every internal node fixes two of three candidate parents before execution; their *actual spike tensors* determine the child's currents. Each artifact is a 32×4 spike tensor. A primary parent alternates across neuron lanes, and $I = \text{primary} + \text{secondary}/8 + 1/16$ ensures the corruption test has observable downstream consequences. This is an untrained lineage test, not a semantic summarization benchmark.

The RTX 4060 Laptop GPU runs FP64 windowed Jacobi with $T = 32$, $L = 8$, $\beta = 1/2$, $\theta = 1$, and $K = 8$. Inputs on the $1/16$ grid and this short dyadic recurrence keep the tested calculations exactly representable; this does not certify arbitrary non-dyadic FP32 inputs. Unresolved tasks replay the full causal reference. We test 10 seeds, logical capacities 1/2/4, and three policies, totaling 90 runs. The dispatcher interleaves task rounds; its Python loop does not execute resident groups concurrently, so capacity is a structural constraint rather than a claim about SM occupancy.

After a leaf commits, its payload is complemented while its independently stored expected checksum remains fixed. Both gated policies reject it before input binding and recompute only that leaf into a new version. All their node artifacts match serial topological evaluation. The ungated control consumes the corrupted tensor, changes two downstream input tensors, and changes 64/128 root spikes in every run (Table 19). The contract does not cover revocation after consumption.

Across 90 runs, 9,840 serialized-state checks and 8,610 actual reentries match uninterrupted task reconstruction, including I, current/previous iterates, boundaries and round alignment. Gated runs average 6.1 full serial replays among 14 attempts including the repaired leaf; the remaining 7.9 attempts finish with an empty adjacent certificate. A separate sweep over 130 clean tasks certifies 20/75/123 tasks at $K = 4/8/12$, while unverified fixed-budget outputs have 740/90/7 wrong spikes respectively out of 16,640 sites. Thus fixed-K output cannot replace the certificate or replay check. The identical matched-baseline result isolates correctness of the shared contract, not a standalone advantage of the DHG scheduling rule.

Internal window-update comparison: delayed state. The rational summary-tree implementation evaluates 240 runs: three leaks, 20 seeds, cyclic/random window selection, and proposal delay zero/two merges. Each $T = 64$ trace has eight length-eight windows. The tree supplies a prefix boundary without a complete global scan. Monotone merges are checked against the causal reference after every event; snapshots include both trees, pending proposals, bounds, and scheduler state. Every run resolves to the exact spikes, including delayed proposals. Cyclic selection is fair with bounded delay; the random runs use fixed seeds and their completion is checked directly.

Block (b) of Table 19 reports the mean operation counts. Selected-window updates avoid already resolved work, but incur summary and tree maintenance. The active-window tree adds 74.95/113.90/282.35 node visits for the three cyclic regimes, plus 71.2/133.2/392.8 charged coordinate readiness checks. Selection no longer rescans the full trace. With random selection and two-event delay, gate counts rise to 230.4/470.4/1764.8 at leaks 0.25/0.70/0.90; the first exceeds the synchronous count. The result supports legal scheduling freedom and exposes its overhead, rather than establishing a uniformly superior policy. The implementation uses exact CPU arithmetic and serial event dispatch; hardware parallelism and validated floating-point tree costs remain open.

Internal artifact-release comparison. A second graph experiment connects the class certificate to execution. A 15-node binary tree has eight source encoders and seven merge nodes. Each node uses a generated fixed signed readout and emits only a binary class. Its two parent classes p_0, p_1 actually alter the bound input: $I_t = I_t^{\text{base}} + (2p_0 - 1)/8 + (-1)^t(2p_1 - 1)/16$. Both sound policies use the same $T = 32$, $L = 8$ exact windowed backend, input selection, class contract, and capacity-limited round-robin queue. One waits for empty U_K; the other tests a class certificate after each round. Twenty seeds at each of two leaks and three capacities produce 240 sound runs. All 3,600 committed artifacts match a causal graph reference the authors wrote, and all 17,331 serialized-state/reentry checks pass.

Every seed/leak instance uses fewer map calls with class release (block (c) of Table 19). Mean logical ticks at capacity four fall from 22.15 to 19.05 and from 52.35 to 45.60. Ticks assume equal map quanta and omit variable certificate overhead; they are not wall-clock latency. An unsafe $K = 4$ policy produces an incorrect intermediate or final artifact in 4/20 and 18/20 instances, and a wrong root in 1/20 and 4/20, respectively. The experiment isolates the effect of the numerical release contract, not a different scheduler or evidence coverage.

H Matched Serving Comparison

The matched serving protocol drives every arm with the same loop, so dependency gates, scheduling and statistics are common and only the numerical execution path changes. Identical SHD test request streams (128 requests, seed 0, deadline factor 3.0, 30% high priority), capacities 4/8/16 and policies FCFS/EDF give six combinations with three repeats per configuration. The two executors this paper compares are ELYSIA-P (batch-parallel serving of independent tasks, with no dependencies between them, under complete-group admission) and ELYSIA (the compound architecture), and each drives each of the numerical backends, so an arm is an (executor, backend) pair. In these runs the ELYSIA arms use the same physical-pool backend as ELYSIA-P. All 90 rows complete 128/128 requests with on-time rate 1.0. The device is an RTX 4060 Laptop, torch 2.11.0+cu128.

A second recorded run of the same protocol on this device is archived but is not used here for a ratio: the archived file records only a two-way producer identity, in its own code-level arm names, and not which ELYSIA execution mode produced each row, so an ELYSIA-versus-ELYSIA-P ratio cannot be reconstructed from it without assuming a row order, and under the alternative assumption the same file gives ratios below one in every cell. We therefore report only the run whose arm labels are recorded. Both records, the frozen sources (`runner_matched_serving.py`, `frozen/`, `run_bench.bat`) and the regeneration notes live in `evidence_external/experiments/results_matched_serving_rtx4060_20260914_formal/`.

Second device (RTX 5090, Blackwell). The same loop, arms and request streams were rerun on a remote RTX 5090 (32 GB, driver 595.71.05, CUDA 12.8, torch 2.8.0+cu128) with five repeats per configuration. All five arms build and run: the JIT fused kernel compiles for `sm_120` once its one MSVC-only flag is applied on Windows only, which is the sole porta-

(a) Joint scheduler/contract control (10 seeds, 90 runs)

Policy	Roots exact	Wrong spikes	Repairs/run	Reentries/run
Contract-matched scheduler (control)	30/30	0	1	98
DHG scheduler	30/30	0	1	98
Ungated negative control	0/30	64/128	0	91
Fixed-K outputs (130 clean tasks)	$K = 4/8/12$: certified 20/75/123; 740/90/7 wrong spikes of 16,640 sites			

(b) Window-update ablation (20 seeds/leak, 240 runs)

β	Sync gates	Selected gates	Summary steps	Tree compositions
0.25	224.0	142.4	222.0	94.05
0.70	544.0	266.4	268.0	151.80
0.90	1907.2	785.6	306.4	357.00

(c) Artifact-release ablation (20 seeds/leak, 240 runs)

β	All-spikes map calls	Class-release map calls	Paired reduction	Unsafe K=4 artifact, root
0.70	55.75 ± 8.16	46.45 ± 7.39	16.58%	4/20, 1/20
0.90	131.75 ± 14.73	114.90 ± 13.30	12.70%	18/20, 4/20

Table 19: Internal ablation: contract, window-update and release controls, three blocks in one table. (a) Scheduler/contract control with real tensor dependencies and one pre-consumption corrupted leaf; both gated policies use identical selected-parent, checksum, scheduling and fallback contracts and their full dispatch traces agree; counts are structural correctness observations; the ungated negative control's 64/128 root spikes are matched by the fixed-K row, where a separate sweep over 130 clean tasks certifies 20/75/123 tasks at $K = 4/8/12$ while unverified fixed-budget outputs carry 740/90/7 wrong spikes of 16,640 sites. (b) Mean operation counts over 20 traces per leak (three leaks, cyclic/random selection, proposal delay 0/2) for fresh cyclic selected-window updates versus full synchronous paired-bound rounds; each gate evaluation also advances one candidate recurrence, summary steps and affine tree compositions are additional work with initialization included, and these counts exclude selection, checkpoint traffic and correctness assertions. (c) Mean and sample SD over 20 seeds; capacities share the same instances and are not additional independent samples; map counts exclude the extra $O(T)$ readout-bound test after each class-policy round; reduction is the mean of paired per-instance reductions; the last column is the unsafe $K = 4$ policy's wrong intermediate/final artifacts and wrong roots. All values are logical work counts or structural correctness observations.

bility change and leaves the timed paths untouched (raw rows, log, runner and build probe: `evidence_external/experiments/results_server_5090_20260915/`). All 150 rows complete 128/128 requests with on-time rate 1.0. ELYSIA with the fused kernel wins all six combinations again and beats ELYSIA-P on every one of the 30 paired repeats (median 1.172, minimum 1.027, sign test $p = 2^{-30}$), and its p95 is below ELYSIA-P's in every cell (0.81–0.96 of it). The margin narrows with capacity, 1.24/1.17/1.04 at capacities 4/8/16, and the same-kernel control below shows that this whole margin belongs to the fused kernel rather than to the architecture (Table 22). Absolute throughput scales with the device (259.65 versus 50.14 tasks/s at capacity 4), so the shipped configuration is the fused one on both GPUs, while ELYSIA's other execution modes (0.64–0.68 of ELYSIA-P) are rejected on both. The full per-cell record of that run is: median throughput (tasks/s, FCFS/EDF) 259.65/263.07 ELYSIA with the fused kernel, 209.56/211.97 ELYSIA-P, 141.39/140.40 ELYSIA with the step-wise LIF under the same packing and 140.49/141.66 ELYSIA with the certified backend at capacity 4; 461.90/457.04, 394.76/391.15, 48.58/48.72 and 46.80/45.39, 263.81/260.19 and 260.28/255.64 at capacity 8; and 734.69/714.20, 704.62/681.09, 81.27/65.91 and 78.71/72.36, 453.33/435.82 and 450.27/434.94 at capacity 16. The per-cell ELYSIA/ELYSIA-P throughput ratios are 1.239/1.241 at capacity 4, 1.170/1.168 at capacity 8 and 1.043/1.049 at capacity 16. For this RTX 5090 campaign, median p95 latency (ms, FCFS/EDF) is 575/571 and 465/460 at capacity 4, 301/303 and 258/260 at capacity 8, and 163/171 and 156/165 at capacity 16 for ELYSIA-P and ELYSIA with the fused kernel; the p95 ratios are 0.81/0.81, 0.86/0.86 and 0.96/0.96. Table 20

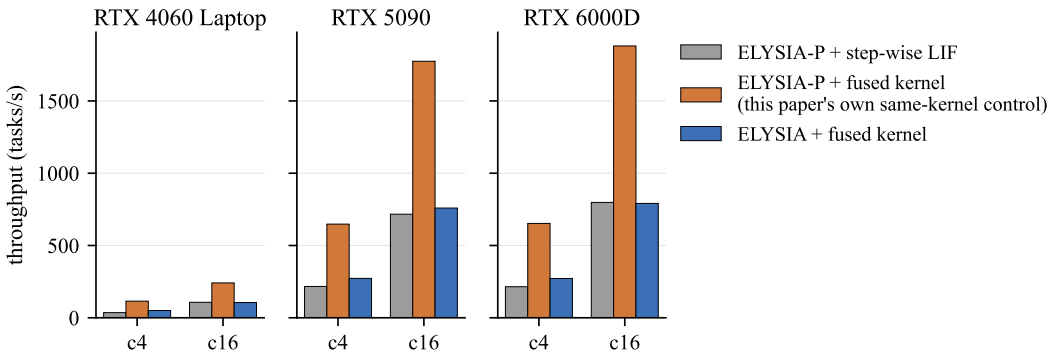


Figure 7: Same-kernel control on three devices (RTX 4060 Laptop, RTX 5090, and the Blackwell RTX 6000-class card this run records verbatim as RTX 6000D; the masked-dense run of Appendix H records the string RTX PRO 6000 Blackwell Server Edition for its own instance), median throughput over the repeats of each cell at capacities 4 and 16 under FCFS, all arms in one process per device on one request stream: ELYSIA-P with its own step-wise LIF, ELYSIA-P with the identical fused kernel, and ELYSIA with that same fused kernel. ELYSIA-P is this paper's own stripped prototype, and here it is the same-kernel control: it holds the numerical kernel fixed so that the difference between it and ELYSIA is the cost of the dependency machinery rather than the contribution of the fused kernel. The fused kernel lifts ELYSIA-P by 2.25–$3.61\times$ on every device, and ELYSIA-P with the same kernel stays ahead of the compound executor, by exactly that machinery's cost at these capacities (Table 22).

Cap.	Policy	ELYSIA-P, step-wise LIF	ELYSIA, step-wise LIF	ELYSIA, fused kernel	ELYSIA, certified backend	ELYSIA/ ELYSIA-P	ELYSIA-P p95 ms	ELYSIA p95 ms
4	fcfs	35.60	23.51	50.14	23.20	1.408	3377	2410
4	edf	38.33	28.19	54.86	25.40	1.431	3156	2196
8	fcfs	73.13	48.58	89.19	46.80	1.220	1628	1345
8	edf	68.66	48.72	89.88	45.39	1.309	1720	1340
16	fcfs	125.66	81.27	135.26	78.71	1.076	915	860
16	edf	102.61	65.91	118.23	72.36	1.152	1135	1002

Table 20: Matched serving comparison: median throughput in tasks/s and median p95 latency in ms over three repeats per configuration, one (executor, backend) arm per column. Throughput ratios divide the ELYSIA-arm median by the ELYSIA-P median; ELYSIA p95 latency is 0.70–0.94 of ELYSIA-P's in the cell medians. The arm these runs label as masked execution is not tabulated as a separate mode: on 2026-09-16 it was shown to execute byte-identical code to the packed arm, because the execution mode only selects the LIF implementation while the batch is always built from the admitted set, so it is a duplicate measurement rather than a second mode. Across the 18 timed repeats one is a reversal (capacity 4, FCFS, repeat 0: ELYSIA-P 46.17 versus the fused arm 29.46 tasks/s, p95 2,439 versus 2,617 ms), and the median repeat-level ratio is 1.256 rather than the median-of-medians 1.264.

reports the separate RTX 4060 campaign, whose absolute latencies are higher because the device is slower; the two campaigns' absolute values are not comparable.

H.1 Internal ablation: masked dense execution against packed admission

One architecture-side claim is measured separately, and both arms of it are this paper's own code, so it costs a component rather than settling a contest. The packed executor keeps only admitted tasks in the numerical batch; the *masked dense executor* keeps every live task and masks the waiting ones with `valid=False`. With identical admission decisions, identical fused kernel, identical inputs and schedule, and identical random weights, the two produce *bitwise identical* outputs on every one of the 20 windows compared. Packing removes $16.1\times$ / $15.1\times$ of the executed neuron-steps at capacity 4 (FCFS/EDF), $4.26\times$ / $4.03\times$ at capacity 16 and $1.364\times$ / $1.369\times$ at capacity 64, and it is therefore the architecture's own effect rather than the kernel's. It does not convert into throughput:

the masked/packed throughput ratio is 1.119 / 1.126 at capacity 4, 0.906 / 0.912 at capacity 16 and 0.963 / 0.965 at capacity 64 — small and non-monotone in the tensor-work ratio. The direct reason is that the fused kernel's device time is flat at 14.6–14.8 μs from batch 1 to 128, so at these sizes the numerical work packing saves is work the GPU was not charging for. The capacity-4 reversal is reported as measured; we do not have an account of it. The run is one process on an RTX PRO 6000 Blackwell (sm_120) with torch 2.8.0+cu128: three arms, three capacities, two policies and five repeats, AB/BA interleaved, 128 requests and 24-step windows.

Cap.	Policy	packed thr.	masked dense executor thr.	masked/packed throughput	batch rows / admitted rows	windows
4	FCFS	244.0	272.9	1.119	16.10	260
4	EDF	244.9	275.7	1.126	15.08	260
16	FCFS	873.5	791.2	0.906	4.26	69
16	EDF	867.4	790.8	0.912	4.03	70
64	FCFS	1673.0	1610.3	0.963	1.364	20
64	EDF	1581.6	1526.3	0.965	1.369	22

Table 21: Internal ablation: masked dense execution against packed admission on one device and one request stream (medians over five paired repeats). The third column is the packed arm, the numerical batch both ELYSIA and ELYSIA-P build when they keep only admitted tasks; the fourth is the masked dense executor; the fifth is their throughput ratio and the sixth is the masked arm's batch neuron-steps divided by its admitted neuron-steps. Both arms complete 128/128 requests, and both are this paper's own code.

Two findings about our own evidence belong in the record. First, the arm the older runs label as masked execution is a no-op label: the execution mode only selects the LIF implementation while the batch is always built from the admitted set, so it is not a masked-execution result and is not cited as one. Second, the older arms runner constructed a fresh backend per arm, so those arms carried independently random weights. That does not affect the published control ratios, for a code-level reason: inside `_admit` the scorer only selects physical pool slots by `topk` and does not choose which tasks run, so the work per window is weight-independent.

Same-kernel control. The ELYSIA-P arm in Tables 20 and 22 drives the package's step-wise LIF (24 Python-dispatched steps per window), so those ratios mix a kernel change with the architecture. Re-running every arm in a single process and a single GPU session—ELYSIA-P with its own step-wise LIF, ELYSIA-P with the identical fused kernel, and ELYSIA with that same kernel, on one request stream—separates the two (Table 22). The fused kernel alone lifts ELYSIA-P by 2.25–3.61× (median over the 16 cells 2.8×); ELYSIA lands at 0.98–1.42× ELYSIA-P with the step-wise LIF (median 1.15×), i.e. it inherits the kernel's gain; and ELYSIA-P with the same kernel is 2.4× faster than ELYSIA (median 0.42×, range 0.31–0.44 over all 16 cells and all three devices). The control is this paper's own dependency-free executor, so that gap measures what the dependency machinery costs to carry at these capacities and 24-step windows; it is not a contest against a third-party system. ELYSIA-P with the step-wise LIF is CPU-bound and the hosts are shared, so its absolute throughput varies by up to 3× across runs on one device while the fused arm is stable (259.7/272.7/270.0 tasks/s at capacity 4 over three campaigns); every ratio above therefore comes from one process on one request stream, and each record stores the host CPU and its load average (`evidence_external/experiments/results_server_5090_20260915/`).

Matched-kernel matrix, and where the architecture's time goes. Table 23 completes the comparison the same-kernel control started: four arms under one protocol, one process, the same fused CUDA kernel, the same SHD request stream and the same scheduling, so the only difference between rows is the named executor or backend. Row (2) is this paper's own stripped prototype, the same-kernel control: it isolates the dependency machinery's cost, so the gap between rows (2) and (3) is what ELYSIA's dependency graph, checkpoints and contracts add. ELYSIA-P is lifted 2.4–3.0× by the fused kernel; ELYSIA with that same kernel reaches 0.43 of ELYSIA-P with the fused kernel, and the certified backend costs a further 1.6–1.9× because it runs the full $K = 8$ window scan with no early release wired into the loop.

Two readings follow, and they point in opposite directions. The architecture's cost is *not* GPU work: the numerical bucket is 5.7× larger in row (1) than in row (3) (6.9× against row (2)), and the device

Device	Cell	n	ELYSIA-P step-wise	ELYSIA-P fused kernel	ELYSIA fused kernel	Kernel gain	ELYSIA/ ELYSIA-P	ELYSIA/ ELYSIA-P fused
RTX 6000D	c4 edf	5	244.6	714.4	289.0	2.92	1.182	0.405
RTX 6000D	c4 fcfs	5	214.9	652.7	272.0	3.04	1.266	0.417
RTX 6000D	c8 edf	5	454.4	1219.2	504.7	2.68	1.111	0.414
RTX 6000D	c8 fcfs	5	456.6	1205.6	504.1	2.64	1.104	0.418
RTX 6000D	c16 edf	5	787.4	1859.4	781.5	2.36	0.993	0.420
RTX 6000D	c16 fcfs	5	798.0	1880.4	791.1	2.36	0.991	0.421
RTX 4060 Laptop	c4 edf	3	42.5	153.6	47.1	3.61	1.108	0.307
RTX 4060 Laptop	c4 fcfs	3	35.6	115.0	50.3	3.23	1.416	0.438
RTX 4060 Laptop	c8 edf	3	60.2	197.5	82.4	3.28	1.368	0.417
RTX 4060 Laptop	c8 fcfs	3	52.1	166.9	65.1	3.20	1.250	0.390
RTX 4060 Laptop	c16 edf	3	93.3	255.2	111.3	2.74	1.194	0.436
RTX 4060 Laptop	c16 fcfs	3	107.1	241.2	105.5	2.25	0.984	0.437
RTX 5090	c4 edf	3	216.6	672.0	276.0	3.10	1.274	0.411
RTX 5090	c4 fcfs	3	216.6	647.9	272.7	2.99	1.259	0.421
RTX 5090	c16 edf	3	716.2	1779.3	753.6	2.48	1.052	0.424
RTX 5090	c16 fcfs	3	716.6	1774.4	759.2	2.48	1.059	0.428

Table 22: Same-kernel control (median throughput in tasks/s, all arms in one process per device, identical request streams). ELYSIA-P is this paper's own stripped prototype, held here on the identical fused kernel as the control that separates the kernel's contribution from the dependency machinery's cost. Columns: ELYSIA-P with its own step-wise LIF; ELYSIA-P with the identical fused kernel; ELYSIA with that same kernel; the kernel gain; ELYSIA over ELYSIA-P with the step-wise LIF; ELYSIA over ELYSIA-P with the fused kernel.

	Executor and backend	cap. 4	cap. 16	p95 cap. 4	p95 cap. 16
(1)	ELYSIA-P, its own step-wise LIF	268.0	891.1	449	128
(2)	ELYSIA-P, the fused kernel (same-kernel control)	788.4	2142.1	154	54
(3)	ELYSIA, the fused kernel	336.7	939.9	359	123
(4)	ELYSIA, the certified backend ($K = 8$)	180.1	572.9	669	200

Table 23: Matched-kernel matrix on one RTX 5090: median throughput in tasks/s and median p95 in ms over five repeats, AB/BA interleaved, one process, capacity 4 and 16 under FCFS. No column is a "best" marker: the throughput winner is row (2), this paper's own dependency-free prototype held on the same kernel, and it wins precisely because it carries no dependency graph, checkpoints or contracts. Rows (1)–(2) and (3) are the paper's same-kernel control; arm (4) is a further (ELYSIA, certified) cell, not the missing ELYSIA-P $\times$ certified corner, which is measured nowhere. Paired ratios: (2)/(1) $= 2.95/2.41$, (3)/(2) $= 0.426/0.438$, (4)/(3) $= 0.536/0.610$, (4)/(1) $= 0.672/0.643$.

kernel itself measures single-digit microseconds (Appendix G.3), so what separates rows (2) and (3) is host bookkeeping in the current Python admission path rather than anything the architecture requires.

How much of that cost is implementation? The decomposition suggests the gap is removable, so we tested that directly. The admission path synchronised the device once per admitted task *before* testing whether that task needed any slots, and rebuilt the affinity matrix every window even when every admitted task already held a complete group. We rewrote it to read the per-task counts back once per window, keep the free-slot count on the host, and build the scorer and free mask lazily. The rewrite decides exactly what the original decided: over 400 randomised states (1–128 tasks, pools of 24–1,536, duplicate and arbitrary admission orders, partial groups, 135 with slot contention) and 433 real serving windows, the owner array, the slot map and the admitted list are bitwise identical, with zero mismatches. Table 25 reports the effect.

The rewrite raises the same-kernel throughput ratio from 0.420 to 0.554 at capacity 16 under FCFS — the cell the pooled figure comes from — and from a four-cell mean of 0.404 to 0.542, with the kernel-gain control unchanged at 2.34–3.01 on both sides. That ratio measures the dependency graph's bookkeeping rather than the way it is written: the same instrumented run shows that with admission removed entirely the ratio would still be 0.72, so the residual $174\,\mu$s per window is work

Executor (capacity 4)	sched-uling	packing-gather	check-point	numer-ical	valid-ation	unattri-buted	window total
(1) ELYSIA-P, step-wise LIF	300.8	119.4	63.3	1340.1	48.9	59.1	1941.4
(2) ELYSIA-P, fused kernel	290.9	102.0	58.0	193.8	46.5	27.5	722.2
(3) ELYSIA, fused kernel	1058.2	113.9	71.8	235.8	48.5	53.0	1583.2
(4) ELYSIA, certified backend	1078.5	125.0	76.6	1490.9	48.8	50.9	2873.2

Table 24: Per-window cost decomposition on one RTX 5090. Each cell is the median over its windows at capacity 4 (1,300 windows, five repeats); the capacity-16 cell carries 345 windows with the same ordering. Every column is a median of a different quantity, so the columns need not - and do not - sum to the total column, which is the median of the per-window totals; the identity that does hold is within each window, where the five named buckets plus the reported residual reproduce that window's wall time to within 1 μs. The residual is reported rather than folded in. Inside row (3), `ELYSIABackend._admit` alone is 707 μs, i.e. 44.7% of the entire window (owner/slot remapping, affinity scoring and per-task `topk` in host Python), and the loop's own priority/age scan costs $\approx$280 μs per window in every row including (1), which never reads those tensors. Absolute microseconds are host-dependent: this run used a Xeon Gold 6459C host, and host dispatch is a single-threaded Python loop, so it scales with per-core clock rather than core count.

Capacity	Window	`_admit` before (μs)	`_admit` after (μs)	Speed-up
4	24	845.4	321.9	2.63×
16	24	1,222.0	740.6	1.65×
64	24	1,947.0	331.2	5.88×
4	96	975.1	729.3	1.34×
16	96	1,586.6	995.9	1.59×
64	96	4,206.1	2,258.6	1.86×

Table 25: Admission cost per window before and after the rewrite, medians over the windows of each cell, one process, FCFS. The rewrite touches only `_admit` and is bitwise behaviour-preserving, so these differences are implementation, not policy.

the architecture performs rather than the way it is written — 56 μs of extra numerics, 16 μs packing, 16 μs checkpointing, 52 μs of non-admission scheduling and the rest unattributed. The measured cost therefore splits into a removable implementation share of about 1.3× and a structural share of about 1.4×, and we report both rather than either. The release-cost figures were also reproduced on this host (fused full trace 0.1116 ms against the published 0.138; certified release 39.53 ms FP32 and 39.55 ms FP64 against 49.9/49.8, ratio 354× against 362×, certification round 3, zero wrong). Table 26 states the protocol behind the published pair.

Per-request arm	median	p95	min	max	n
Fused full trace (ms)	0.1379	0.1464	0.1353	3.243	128
Certified release, FP32 (ms)	49.89	104.17	16.67	168.90	128
Certified release, FP64 (ms)	49.83	105.23	16.54	172.64	128

Table 26: The study behind the paper's 49.9 ms and 0.138 ms statement in Appendix F.1. One RTX 5090, torch 2.8.0+cu128, one process, no co-tenant. The arm shares the gate set of Table 14: the same 2,264-sample set, 86.31% early release, zero wrong releases, zero never certified. The timing boundary is the complete per-request service call, including input preparation and the enclosure rounds, not kernel time; the fused row is one full-trace request on the same stream. Certification round is 3 in both precisions, so the certificate is right about the class and costs 362× this paper's own fused trace in median latency. Reproduced by the authors on a Xeon Gold 6459C host the ratio is 354× (39.53/39.55 ms against 0.1116 ms) with the same certification round and zero wrong releases: the absolute milliseconds are host-dependent while the ratio moves by only 2% across hosts.

Three certification costs, three questions. This paper reports certification at 362× the fused trace per request (Table 26), at 1.6–1.9× the throughput of the same loop on the fused kernel (Table 23), and at 1.95× per task-window on the producer side inside a mixed-consumer batch. These are not

three estimates of one quantity: the first is a single request's release latency, the second is steady-state serving cost when every window runs the certified scan, and the third is the scan's overrun on one producer while its consumers idle. They differ by design and are neither multiplied nor averaged anywhere in this paper; the only quantity common to all three is the certified scan itself, and what each of the three costs gives is the same thing — a released class that equals the reference.

Independent request streams, and what the intervals cover. Every serving number above comes from one request stream. We therefore rebuilt the comparison on 18 independently generated streams (arrival spread $\in \{4, 24, 64\}$ crossed with a high-priority fraction $\in \{0.15, 0.45\}$, three seeds each) plus one anchor stream that reproduces the original protocol verbatim, one process per stream, five repeats per configuration, arms and capacities AB/BA-interleaved. Weight fingerprints confirm the projection and readout are byte-identical across arms, so only the named component differs.

Ratio (median over 18 streams)	Capacity	median	95% interval	excludes 1
ELYSIA (fused kernel) / ELYSIA-P (step-wise LIF)	4	1.249	[1.225, 1.298]	yes
	16	1.124	[1.033, 1.196]	yes
ELYSIA (fused kernel) / ELYSIA-P (fused kernel)	4	0.479	[0.432, 0.535]	yes (below)
	16	0.650	[0.587, 0.694]	yes (below)
ELYSIA-P (fused kernel) / ELYSIA-P (step-wise LIF)	4	2.632	[2.290, 2.935]	yes
	16	1.725	[1.641, 1.848]	yes
ELYSIA (fused kernel) / ELYSIA (certified backend)	4	1.734	[1.699, 1.779]	yes
	16	1.555	[1.428, 1.668]	yes

Table 27: Stream-level intervals (paired bootstrap over the 18 streams, 20,000 resamples, statistic = median); all ratios are of one executor and backend against another. One RTX 5090, one process per stream, FCFS, 24-step windows, medians over five repeats, AB/BA interleaved. The replication unit is the stream, not the repeat; the interval covers the stream population of this generator family and not resampling of the sample set, weights, contexts, capacities or hosts. These are the 18 generated streams, not the single seed-0 stream behind Tables 22 and 23: burstier arrivals leave the capacity-16 prototype less utilised here (999.1 against 1,774–1,880 tasks/s), so ratios may not be carried between those tables. Per-cell median throughput over all 19 streams (the 18 factorial streams plus the anchor): at capacity 4, ELYSIA-P 227.5, ELYSIA-P with the fused kernel 583.1, ELYSIA with the fused kernel 290.2, ELYSIA with the certified backend 164.9 tasks/s; at capacity 16, 568.3 / 999.1 / 676.4 / 428.8.

Three consequences. First, the direction of every earlier conclusion survives but the pooled number does not: the anchor stream alone gives 1.235 at capacity 4 and 1.038 at capacity 16, so the ratio is capacity-dependent and a single pooled figure such as the earlier $1.264\times$ is not reproducible across streams. Where each executor keeps its own kernel, rather than sharing one, the contrast is 1.249 $[1.225, 1.298]$ at capacity 4 and 1.124 $[1.033, 1.196]$ at capacity 16 — the first two rows of Table 27 — and on the anchor stream alone ELYSIA runs at $1.235\times$ and $1.038\times$ ELYSIA-P. Second, the earlier claim that fused p95 is lower in every cell holds for all 18 streams at capacity 4 but only 16 of 18 at capacity 16 (mean ratio 0.955, interval crossing 1). Third, and most usefully, the protocol itself was under-powered: re-running the earlier test stream by stream with three repeats declares the capacity-4 ratio significant in only 9 of 18 streams and the capacity-16 ratio in 4 of 18, while the stream-level interval over the same data clearly excludes 1. Treating the 90 repeat-level ratios as independent narrows the interval by 1.5–2.9$\times$ (the variance-component intraclass correlation, median across cells, is 0.003), which we report rather than overstate. Host thread budget moves these ratios by 1.2–1.5$\times$ on the same machine, so only same-host paired ratios are comparable.

Consumer release through a real downstream workload. The certificate's practical claim is tested with one shared producer and two consumers on the same 128 requests: a class-only consumer that may start at the class contract and a full-trace consumer that waits for the trace. Both configurations run the same graph, the same scheduler, capacity 4 and FCFS; they differ only in the producer's numerical backend, and at an 8-step window the certified scan's spike trace is *bitwise* identical to the fused backend's (0 of 841,728 spike positions, and identical class assignments, readout scores, publish windows, window executions and neuron-steps; the membrane potential is not bitwise equal, with a maximum absolute difference of 3.7×10^{-8}), so the two configurations execute the same numerical graph and differ only in what they charge for it (Table 28).

Event (seconds from the producer's first window)	fused	certified	paired difference
class contract publish	0.8996	1.7545	+0.8527
full-trace contract publish	0.9112	1.7800	+0.8667
class-only consumer start / finish	0.8996 / 0.8997	1.7545 / 1.7546	+0.8527 / +0.8528
full-trace consumer start / finish	0.9112 / 0.9112	1.7800 / 1.7800	+0.8667 / +0.8667
root completion	0.9112	1.7800	+0.8667
class-only consumer head start	0.0116	0.0257	+0.0139

Table 28: Mixed-consumer timeline (medians of five repeats, AB/BA interleaved, 128 requests, capacity 4, FCFS; the replication unit is the repeat). The paired difference column is the median of the five per-repeat pairwise differences, so it is not in general the difference of the two preceding columns; the five pairwise differences are positive in 5 of 5 repeats for every row. Total logical work is identical in both configurations (747 scheduling windows, 2,945 task-window executions, 565,152 logical neuron-steps); the certified scan executes 4,521,216 kernel neuron-steps against 565,152 for the fused scan, $8.00\times$ as many. The consumer service times themselves have intervals straddling zero.

The result is negative and holds for three independent reasons. The logical release point is *identical* in the two configurations: the class contract is content-determined and fires at block 4 (step 64, when the last readout block becomes observable), and it cannot fire earlier, because one unobserved block can still contribute 192 in the frozen readout's integer score units and no single-block prefix can overcome that; early release therefore gives no relative advantage on this workload, with this readout and in this $K = 8$ loop. A free certificate does not change that: an oracle arm that runs no certificate at all and stops each task at the round a previous certified pass recorded still saves only 1.6% of task-rounds ($2{,}016/2{,}048$; 5.3%, $1{,}940/2{,}048$, with the tighter interval bound) and keeps every dense row ($1.000\times$), and it still costs $1.43\times$ the plain loop, because 227 of 256 tasks never certify before the $K = 8$ budget is spent. The mechanism is net-negative by construction at the workload measured here — this readout, this $K = 8$ loop, this launch-bound regime — and not because certification itself is expensive. What bounds it is a cost paid before any hit rate applies: defusing the K-round loop, without which a task cannot be stopped at all, costs 1.30–$1.46\times$ by itself, and the certificate's per-round ops cost a further 2.4–$3.6\times$. At $T = 24$, where 37.9% of rounds are genuinely skipped, the net wall clock is still $+156\%$ to $+230\%$, and no crossover appears up to batch 64: in this launch-bound regime, and on the workloads measured here, the ceiling on early release is negative however often the class contract fires. The downstream step is trivially small: 0.148 ms (fused) and 0.170 ms (certified) of consumer work against producer spans of 0.910 s and 1.777 s, i.e. 0.016% and 0.010%. The cost sits on the producer side: the certified scan is $1.95\times$ slower per task-window and 98.5% of that overrun is already incurred by step 64. Two *derived* (not measured) crossovers follow, and they answer different questions. Holding the certified producer's overrun $\Omega = 0.868$ s against the class consumer's own service time $c = 0.148$ ms gives $\Omega/c \approx 5{,}860$: the class-only downstream work would have to be that much heavier before early release repaid the certification latency. Against the certification latency itself, $\Lambda = 0.855$ s shared over 128 requests, the break-even is $\Lambda/128 = 6.68$ ms of downstream work per request, which is $45 \times c$; the two ratios differ because Ω is the producer overrun while Λ is charged over the whole batch.

Under the most favourable truncation assumption the certified producer still finishes 0.845 s later. Certification's defensible contribution is therefore exact stopping plus compositional correctness (bitwise-identical spike traces and identical root artifacts across two numerical backends), not latency. One provenance caveat: for this path the frozen int64 readout was fitted on the first 1,024 official test samples and measured on the disjoint samples 1,024–2,264 (1,240 samples; held-out accuracy 0.694, Wilson interval [0.668, 0.719], against 0.05 chance), because the official training archive was not obtainable at measurement time; the timing conclusions do not depend on the readout.

Revision density, and the latency from correction to a valid result. The same harness sweeps the revision density d (expected revisions per root completion) rather than the width alone, and times the interval from a revision to a valid root artifact again, split into invalidation bookkeeping and re-execution (Table 29).

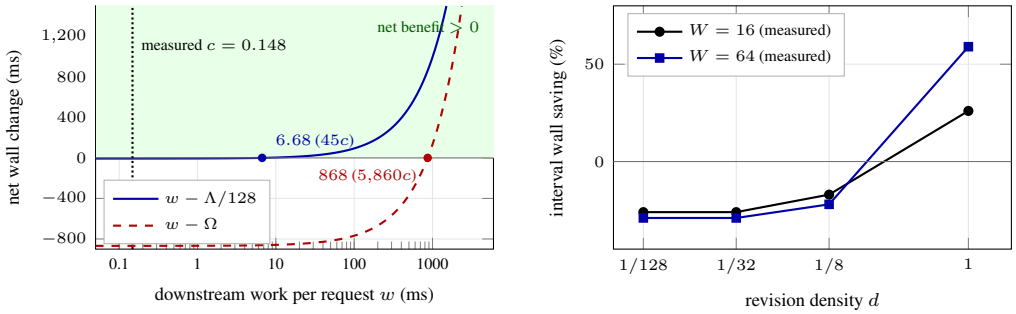


Figure 8: Where each guarantee stops paying, on one axis per mechanism. *(a)* Certification, against measured consumer work. The only measured operating point is the class-only consumer's own service time $c = 0.148$ ms; the derived break-evens are 6.68 ms when the certification latency is charged over the whole batch of 128 requests, and 868 ms when the producer's own overrun $\Omega = 0.868$ s is charged to a single consumer. The two thresholds differ by a factor of 130 because they charge different quantities; both come from measured spans (Λ and Ω), neither is a measured crossover. *(b)* Selective repair, measured. Interval wall saving against revision density d at two widths; the saving is positive only at $d = 1$ and has crossed zero by $d = 1/8$: on this harness, at these two widths and these per-revision costs, selective repair pays only when revisions are frequent. Points are the four realised densities of Table 29; the line is drawn to guide the eye, not fitted.

W	d	replayed work ratio	paired wall ratio per revision	correction latency repair	paired difference recompute−repair	recompute slower
16	1	16×	1.3933	0.2447 ms	+0.0962 ms	561/567
16	1/8	16×	1.3780	—	—	72/72
16	1/32	16×	1.3355	—	—	41/41
16	1/128	16×	1.3454	—	—	41/41
64	1	64×	2.4356	0.3442 ms	+0.4591 ms	**566/567**
64	1/8	64×	2.1637	—	—	72/72
64	1/32	64×	2.1622	—	—	41/41
64	1/128	64×	2.1307	—	—	41/41

Table 29: Revision density against graph width: replayed-neuron-step ratio, the per-revision paired wall ratio (recompute over repair, median over the paired revisions of the cell), and the measured interval from a revision to a valid root artifact (nine repeats, AB/BA interleaved, zero failures, one ground-truth equality check per repeat and nine per configuration). The realised densities are 0.0319 at $d = 1/32$ and 0.0080 at $d = 1/128$, so the density axis does vary; d is the requested rate and the harness raises the root count to keep the observed revision count per repeat comparable. Recompute is the ELYSIA-P behaviour on a revision: a lineage-free executor cannot narrow the replay set. Repair replays exactly the affected subtree, so the work ratio equals W at every density; the wall ratio is dominated by the width rather than by the density (1.34–1.39 at $W = 16$ and 2.13–2.44 at $W = 64$ across all four densities), because each revision costs roughly what it costs regardless of how often it happens. The paired difference is recompute minus repair, so a positive value is the time repair saves per revision; the last column counts the paired revisions in which recompute was slower, with a one-sided sign test at $p = 9.4 \times 10^{-158}$ ($W = 16$) and $p = 1.2 \times 10^{-168}$ ($W = 64$) at $d = 1$.

Two derived quantities locate when saved replay work outweighs recovery bookkeeping. The repair arm's own invalidation bookkeeping is 0.0110 ms of its 0.2447 ms correction latency at $W = 16$ and 0.0110 ms of 0.3442 ms at $W = 64$, i.e. 4.5% and 3.2% of repair's own latency, or 3.3% and 1.4% of the recompute latency it is compared against. Charging the bookkeeping to repair, the measured replay saving on the affected subgraph is 29.8% at $W = 16$ (0.2303 ms of re-execution against 0.3279 ms of recompute) and 58.3% at $W = 64$ (0.3277 against 0.7859), against the ideal $1 - 1/W$ of 93.75% and 98.4% that holds only when repair's re-execution is exactly $1/W$ of recompute's. The gap between work and wall is far wider than those percentages suggest: recompute re-executes 16× (64×) the neuron-steps of repair but takes only 1.39× (2.44×) the wall time, so the wall benefit is a factor 11.5 ($W = 16$) and 26.3 ($W = 64$) smaller than the work benefit. Absolute latencies on this shared host vary between runs of the same binary (an earlier run gave a repair correction

latency of 0.294 ms and 0.350 ms at the two widths, 17% and 2% above the values tabulated here), so the ratios, not the absolute times, are the transportable quantity. At density 1 the exact discrete interval-level saving $m(W-1)/((R+m)(W+1))$ is 44.1% ($W=16$) and 48.5% ($W=64$); at density $1/32$ the same quantity is 2.7% and 3.0%, and at $1/128$ it is 0.70% and 0.78%. The earlier 0.7–2.4% figure came from a different, non-uniform graph, and this sweep's values bracket it one density step lower rather than reproducing it; both denominators (replay versus whole interval) are reported here. The two axes therefore drive different denominators, and the density sweep locates the crossover. Width sets the replay-level saving, because one revision re-executes one branch window either way: $(W-1)/W$, i.e. 93.75% and 98.4%, at every density. Density sets the complete-interval saving, which is near-linear in d and saturates in W: the deterministic interval neuron-step saving is 46.5%/48.8% at $d=1$, 10.4%/10.9% at $1/8$, 2.9%/3.0% at $1/32$ and 0.74%/0.78% at $1/128$ ($W=16$/$W=64$), matching the discrete form to within 2.4 points in every cell. Since a clean completion costs both arms the same work but still charges repair 0.0110 ms of controller bookkeeping, and only a $1/d$ fraction of completions is revised, that fixed cost eventually exceeds the saved replay: the measured interval wall saving is about $+26\%$/$+59\%$ at $d=1$ and turns negative, -17%/-22% at $d=1/8$, -26%/-29% at $d=1/32$ and -26%/-29% at $d=1/128$ ($W=16$/$W=64$), so on this loop, at these two widths and these per-revision costs, selective repair is a net wall-clock loss at $d=1/8$ and below, and repays that bookkeeping only where revisions are dense enough to amortise it — the $+26\%$/$+59\%$ at $d=1$ measured above. The signs reproduce in the earlier run of the same binary, whose $d=1$ interval wall savings were $+28\%$/$+61\%$. We nevertheless do not quote an interval-total wall *ratio*: at $d=1$ a cell holds a single interval whose first completion carries a warm-up transient, and that ratio moved by up to 74% between runs of the same binary, so the per-revision paired ratio in the table is the reproducible one.

Revision cost scales with the affected subtree, not the root. The same harness answers the complementary question: what does selective repair pay when a revision is frequent? We build a compound root of W independent branch tasks plus one fusion task, revise one uniformly chosen branch after the root completes, and time both lineages: *repair* invalidates the consumed descendants of that branch and re-executes only them, while *recompute* re-executes every branch and the fusion. Both arms run the same fused kernel, the same batching and the same inputs, and their root artifacts are bitwise identical in every timed revision (864 revisions, 0 mismatches). Recomputation is what ELYSIA-P must do here, having recorded no lineage.

W	replayed neuron-steps	recompute/ repair work	graph tasks	wall ratio (batched)	wall ratio (per-task dispatch)
2	2,048	2×	1.5×	1.052	1.752
4	2,048	4×	2.5×	1.086	3.212
16	2,048	16×	8.5×	1.431	12.696
64	2,048	64×	32.5×	2.124	37.733

Table 30: Lineage repair against whole-root re-execution as a function of graph width (median of 108 paired revisions per configuration, AB/BA interleaved). Repair replays exactly one branch window at every width, so the work ratio grows with W (columns 3–4, clock-independent); the wall ratio is measured under the laptop GPU's idle-clock spread and moved between 1.94 and 2.12 at $W{=}64$ across four runs of the same binary (the delivered run, archived with this appendix, is the 2.12). Two readings matter. First, the wall gain is *host-side*: at $W{=}64$ each extra branch costs $\approx 19\,\mu$s while the kernel-only difference between batch 1 and batch 64 is inside the noise, so the saving is the 63 per-branch host pipelines, not GPU arithmetic. Second, this does not reproduce the paper's earlier 0.7–2.4% complete-interval figure: that number is set by the revision *density* ($1/32$ there, 1 here), and at density $1/32$ the same analysis gives $\approx 3\%$ even at $W{=}64$.

Kernel gain grows with the window length. Holding the request stream fixed and sweeping the window length and the capacity on one device separates two curves: the fused kernel's advantage over the step-wise LIF grows monotonically with the window (from 1.71–$3.08\times$ at 24 steps to 4.54–$8.83\times$ at 96 steps), because the step-wise path pays a Python dispatch per step, while the architecture's own cost stays flat at 0.315–0.450 of ELYSIA-P with the same kernel across all nine configurations. Longer windows and higher concurrency therefore do not amortise the architecture; they only widen the kernel's margin (Figure 10).

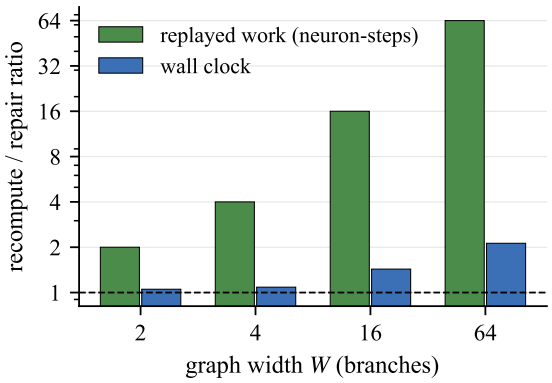


Figure 9: Replayed work and wall clock of repair relative to whole-root re-execution against graph width (Table 30). The work ratio tracks the width; the wall ratio saturates far below it because the saved work is nearly free on the GPU.

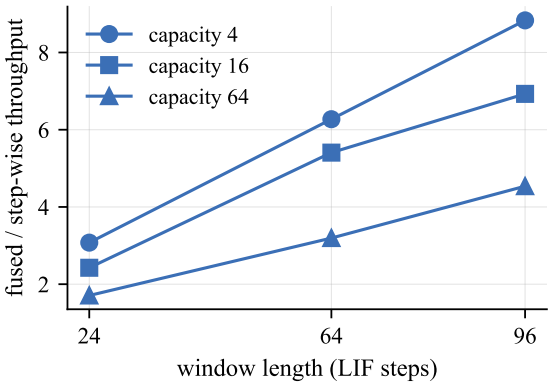


Figure 10: Fused-kernel gain over the step-wise LIF as a function of window length, at three capacities, on one device and one request stream.

I Planned Experiments

The completed studies above cover two learned fixed graphs, one public-workflow capacity set and the exact dyadic CUDA conditions. Table 31 lists the experiments a reader would need in order to carry each remaining claim past its present scope. *No row of it is measured in this paper*, no row is cited anywhere else in the paper, and the result column is deliberately empty. Each extension reuses the frozen protocols and evidence tree of this paper; protocols and reserved table formats are archived in the companion package (`roadmap_planned_experiments.md`).

I.1 Per-request cost of the exact-dyadic class scan

The certification costs in §5.1 and Appendix H come from two different implementations of the same release contract, and the gap between them is two orders of magnitude. The 49.9 ms per request is the outward-rounded FP32/FP64 enclosure of Appendix G.8. The exact-dyadic two-limb CUDA scan of Appendix G.10 implements the same conditions in exact integer arithmetic, and Table 32 measures it directly.

Two checks run by the authors tie this scan to the numbers already in the paper. Its mean certification rounds are $8{,}557/2{,}264 = 3.780$ per request at one view and $9{,}418/2{,}264 = 4.160$ at three, which are the single-view and shared-producer reductions quoted in §5.1; the full-spike arm's 5.874 and 13,299 reproduce as well. Every timed output equals the arbitrary-precision reference.

The consequence for the release argument is arithmetic. Break-even is one extra producer request-time of downstream work, so it is 6.68 ms per request on the enclosure path and 0.149 ms on the dyadic path, against the consumer's own 0.148 ms measured in Appendix H. The enclosure therefore

Ref	Experiment	Extends	Result
P1	An external, hardware-measured baseline at matched accuracy: an independently authored SNN runtime (snnTorch CUDA neuron, Lava, or a dedicated GPU SNN engine) serving the same request stream on the same device	the serving comparison of §5.5, which is against this paper's own prototype only	**done**: two genuinely independent runtimes, `snnTorch` and `SpikingJelly`, both at $1{,}742/2{,}264$; the other two paths measured in the same study — the authors' own fused kernel and the authors' own PyTorch LIF — are ours, not external (Appendix I.9)
P2	The throughput–guarantee frontier at a stated operating point, including a no-guarantee fast path and a mixed-traffic measurement	the opt-in reading of the $0.42\times$ cost	—
P3	Certification outside the dyadic conditions: $\beta \in \{0.8, 0.9, 0.95\}$, non-dyadic inputs, and a recurrent or multilayer model	the scope of §4	**done in part**: $\beta = 19/20$, non-dyadic, 512 neurons, $T = 100$, all 2,264 classes released with zero wrong (Appendix I.3); a recurrent or multilayer model remains open
P4	A scale ledger: $\geq$ 512 neurons, $T \geq 100$, 128–1,024 live tasks, with per-window time budget, occupancy and per-task state bytes	the capacity regime measured here ($L \leq 64$)	**done in part**: 512 neurons and $T = 100$ (Appendix I.3), and occupancy at the scanned cells (Appendix I.5); live-task count and per-task state bytes remain open
P5	Per-request consumer start latency for the mixed class-only/full-trace producer scenario	the consumer-release result of §5.1	—
P6	A published higher-accuracy SHD configuration ($\geq$ 90%) trained into the compound graph and certified	the 57–68% models used throughout	**done in part**: the published-configuration DCLS trunk, certified at 0.92182 with 2,007 of 2,264 released and 0 wrong releases (the subsection following Appendix I.3); training it *into* the compound graph remains open
P7	A second application domain for the repair comparison, and learned graphs of 8–16 subtasks	the two learned fixed graphs of §5.2–5.3	—
P8	Recovery under timing jitter, concurrent stale writers and deadline misses	the synchronous injected faults of §5.3	—

Table 31: Planned experiments, listed so that the boundary of the present evidence is explicit. Rows marked *done* were added after this table was first written and are reported in the cited appendix; every unmarked row is not measured anywhere in this paper, its result column is empty by construction, and no number in the paper derives from it.

needs $45\times$ the consumer's work to pay for itself, while the dyadic scan essentially breaks even. Both figures are on the exact same conditions and the same model; only the arithmetic differs. Which

Configuration	Fused	Class scan	Ratio
Per request, one sample (views 1)	0.137	0.286	2.08×
Per request, one sample (views 3)	0.152	0.312	2.05×
Batch of 2,264 (views 1)	0.442	2.131	4.82×
Whole split as 2,264 single requests	0.156	0.297	1.90×

Table 32: Median milliseconds for the fused full trace and for the exact-dyadic class scan, RTX 4060 Laptop, timer around the backend call so host dispatch, every certificate kernel, per-wave completion syncs and the class/release readback are inside the boundary. The per-request rows pool 210 observations per mode over two runs; the batch row is the archived run. Ratios are the portable quantity: absolute milliseconds on this host move with GPU clock state, and a same-session rerun of the unmodified original runner reproduces the batch ratio (4.88 against the archived 4.82). The $4.82\times$ batch figure keeps its own device and boundary: RTX 4060 Laptop, timer around the backend call, which is the archived boundary the paper's $0.442/2.131$ ms pair was taken on. On an RTX PRO 6000 Blackwell under an exclusive block the same comparison gives $11.9\times$ on that device-event boundary and $7.8\times$ at the vote-inclusive boundary of Table 5, because the fused kernel collapses to $48.6\,\mu$s there while the scan still pays twelve sequential rounds of launches and per-wave host synchronisations. The per-request $2.08\times$ is the stable one and reproduces at $2.23\times$.

implementation a deployment chooses is a precision decision, not a certification decision, and the paper reports both rather than either.

I.2 An exact one-step release rule, and what it gives

The rule this paper ships widens a sound interval by one step of the reset-to-zero recursion and repeats. That widening is sound but not tight: at $u = \theta$ the cases of Proposition 11 overlap, so a coordinate can stay unresolved even where the reachable membrane is already a point. The hull rule removes that slack. The membrane at time t depends on a candidate history only through the last spike at or before t, so the attainable set of that index over a box $l \le q \le h$ is exactly

$$A_t(l,h) = \{\ell \in [\max(\rho,0),\, t] \,:\, h_\ell = 1\} \cup \{-1 \text{ if } \rho = -1\}, \qquad \rho = \max\{j \le t : l_j = 1\}, \quad (6)$$

because ℓ is attainable exactly when it can be one and nothing after it is forced to be one. The hull rule takes $\min$ and $\max$ of $S_{t-1}(\ell)$ over A_{t-1}, where $S_t(\ell)$ is the membrane at t of a neuron whose last reset was at ℓ, and updates the endpoints from $H(\beta \cdot \mathrm{lo} + I_t - \theta)$ and $H(\beta \cdot \mathrm{hi} + I_t - \theta)$. Two machine checks fix what that gives. A_t matches brute-force enumeration in $336/336$ boxes, and the hull rule's single step matches the exhaustive Φ_{rz} range over the box in $88/88$.

Proposition 10 (The hull rule attains the exact one-step box refinement)**.** *Call a rule* sound one-step box *if its state is a box $l \le q \le h$ and it returns endpoints containing the reachable spike range after one application of the reset-to-zero recurrence to that box. Then the hull rule returns exactly that range, and its box is contained in the box of every sound one-step box rule.*

Proof. The spike at t depends on the candidate history only through the last index $\ell \le t-1$ with $q_\ell = 1$, because earlier spikes have been reset away by then: $S_t(\ell)$, the membrane at t of a neuron last reset at ℓ, is fixed by ℓ and the currents. The set $A_{t-1}(l,h)$ of (6) is exactly the set of ℓ that some q in the box attains, since ℓ is attainable precisely when $h_\ell = 1$ and no j with $\ell < j \le t-1$ is forced to one. So $\min_{\ell \in A} S_{t-1}(\ell)$ and $\max_{\ell \in A} S_{t-1}(\ell)$ are the exact attainable membrane range, and $H(\beta \cdot \mathrm{lo} + I_t - \theta)$ carries its endpoints to the exact attainable range of the spike at t. The hull rule evaluates these two quantities. A sound one-step box rule must enclose that range, because its endpoints are sound for every q in the box; hence its box contains the hull rule's. The $336/336$ and $88/88$ enumerations check the two premises computationally. □

The class is narrow, and that is what the next paragraph is about: our sequential-sweep rule keeps a membrane enclosure beside its box and so sits outside it. On exact-rational populations at $\beta = 19/20$ it certifies 84.8% of coordinates at $T = 8$, 89.8% at $T = 16$ and 92.9% at $T = 24$, against 74.8%, 78.5% and 84.0% for the shipped parallel rule, and it dominates that rule in $4{,}590/4{,}590$ coordinate-rounds with zero soundness violations for either. That dominance is aggregate rather than coordinate-wise. On the deployed recurrence over 24 rounds, 93 of $2{,}782{,}003{,}200$ coordinate-rounds (3.34×10^{-8}) go the other way. The reason is scope: the exactness is proved for rules whose

state is the box, while our sequential-sweep rule also carries the membrane enclosure of its own chain. Aggregate dominance is still what the rule is for — it releases every class instead of 499 of 2,264 in 36 rounds instead of 60 — and the 93 coordinate-rounds are the cost of comparing a box-state rule against one that carries an enclosure as well. A correction is recorded because a check caught it: the first version of the hull rule used the interval $[L_t(l), L_t(h)]$ of last-spike indices, which is wrong because h may have zeros strictly inside it, and dominance failed in 2 of 4,590 rounds until (6) replaced it.

On the deployed recurrence the cost and the return are both measured (Table 1). The hull rule spends $1.31\times$ more per round, 8.84 s against 6.73 s, because (6) does more work per round than a single widening step. It returns that in rounds: 36 instead of 60, a median certification round of 12 under the hull rule instead of 54 under the shipped sequential-sweep rule, a p90 of 20 where our sequential-sweep rule's median is 54, every one of 2,264 classes released instead of 499 of them, and it reaches all of them on 318.2 s of round budget against our sequential-sweep rule's 403.8 s under the same exclusive block. Both rules hold 0 wrong pins, 0 wrong released classes and 0 lower or upper enclosure violations, and both pass the same gates first: the deployed FP32 accuracy equals the float64 reference at $1{,}742/2{,}264 = 0.769434628975265$, with 0 class and 0 spike disagreements over 115,916,800 coordinates. The hull rule is an improvement to our own release rule and is costed as an internal comparison; it is not a comparison against anybody else's method, and nothing here weakens Proposition 9, which is scoped to the agreement channel of Theorem 3.

Model	T / hidden / τ	Released, shipped	Released, hull	Median, shipped	Median, hull
Control	100 / 512 / 20	0	2249	—	12.0
Leak	100 / 512 / 10	306	2264	26.0	7.0
Width	100 / 256 / 20	35	2264	29.0	11.0
Horizon	256 / 512 / 20	0	2260	—	11.0

Table 33: Internal ablation: our sequential-sweep rule against the hull rule on four models at the same round budget $K = 30$, on the official test split of 2264 samples. Both rules see the same frozen currents, the same proved prefix and the same margin test, and the comparator is this paper's own sequential-sweep rule. The four models reach deployed fp32 accuracies 0.7690, 0.7120, 0.7359 and 0.8454 in table order. The hull rule releases at least as many roots as our sequential-sweep rule on 4 of 4 models, and its pinned set contains our sequential-sweep rule's on 3 of 4: on the 256-neuron model our sequential-sweep rule pins 31 coordinates the hull rule leaves unpinned, the same aggregate-rather-than-coordinate-wise scope already recorded above. The fp64 soundness check passes on three models. On the $T = 256$ horizon model it is flagged by *one* coordinate out of 296,747,008 — sample 323, step 19, neuron 112 — whose membrane arrives half an fp32 ulp below θ (0.99999994 against 1.0), so fp32 withholds a spike the exact recurrence fires and the box cannot contain both. The near-tie the 512-neuron model records above sits one float32 ulp from θ; this one is the midpoint case, at exactly half an ulp. What the flag counts is that one unresolved coordinate held across rounds, not eleven independent violations: it stays unresolved from round 20 to round 30 under both rules, which is 11 upper-enclosure coordinate-rounds each against the float64 recurrence. The two rules are not equally exposed by it. The sequential-sweep rule releases no root within this budget, so it never publishes that coordinate and passes the gate at and after release. The hull rule releases 2,260 and carries the violation at or after the round that published it; on the deployed fp32 basis the same coordinate puts the hull rule in violation from round 2, and the hull rule is the only one of the two rules that pins it — in 18 of its 40 violation rows, to the bit fp32 gets wrong, where our sequential-sweep rule pins it in none. None of that is a claim the gate is making: 0 class disagreements and 0 wrong released classes hold for either rule on either basis, and the hull rule's coverage advantage holds on 4 of 4 models. The hull rule is not cheaper: it spends $1.31\times$ per round on the control model, 8.84 s against 6.73 s at $T = 100$, and $1.67\times$ on the horizon model, 29.23 s against 17.47 s at $T = 256$. What it returns is coverage and rounds to coverage, on every model.

What transfers, and what that does not claim. The rule is stated over a box state, its exactness is proved for that class, and the table above shows it improving coverage on four models that differ in leak, width and horizon under the deployed reset-to-zero convention. That is a property a reader can check against their own model; this paper does not measure, and does not claim, what anyone else will do with it. Two limits travel with the result. The first is the soundness flag on the $T = 256$ model above, which is a half-ulp tie both rules meet rather than one the hull rule introduces, and

which leaves both rules releasing 0 wrong classes there; what differs between them is only which rule publishes the coordinate. The second is pin-set containment, which is a property of the attainable-last-spike set rather than a theorem and is falsified at $H = 256$, exactly as the qualifier above records for the deployed recurrence.

I.3 The certificate at 512 neurons

Every certified study in this paper uses a model that meets the exact-dyadic conditions. This one instantiates the same certificate on the 512-neuron SHD model of Table 16, which reaches $1{,}742/2{,}264$ (76.94%) and whose leak, $\beta = 1 - 1/\tau = 19/20$, is *not* dyadic: $19/20$ carries a factor five in its denominator, so β^t is not a dyadic rational and the exact-dyadic conditions of Appendix G.8 does not apply. The enclosure is therefore outward-rounded FP32/FP64 over the exact FP32 currents, with an exact integer readout.

	FP32 (CUDA)	FP64 (CPU)
Roots released	2,264/2,264	2,264/2,264
Never certified	0	0
Wrong at release, vs exact reference	0	0
Wrong at release, vs deployed class	0	0
Released class accuracy vs label	76.94%	76.94%
Median certification round, of $T = 100$	11	11
Pinned bits audited	1.16×10^8	1.16×10^8
Enclosure violations, every round	0	0

Table 34: The certificate on a 512-neuron 76.94% model with a non-dyadic leak, against the instance of Table 14, which releases 83.75% at $\beta = 1/2$. Release is faster as a fraction of budget here (11.0% of 100 against 20.8% of 24) despite $5.6\times$ the width and a slower leak. The rule behind these 11 rounds is the reset-to-zero interval extension of Proposition 11 with Corollary 12, not the hull rule: the hull rule reaches all 2,264 at a median round of 12 on the same recurrence (Table 1). The released class equals the model's own reference prediction on every root, so certification changes no prediction and the model's task accuracy is unchanged.

The reproduction is gated: the deployed FP32 forward pass gives $1{,}742/2{,}264$ exactly, and its argmax agrees with the exact reference on all 2,264 roots. Release is audited after every round over $2{,}264 \times 100 \times 512$ coordinates in both precisions with zero lower and zero upper violations; an exact-rational falsification test in `Fraction` arithmetic over 5,120 adversarial systems finds none; a second implementation written by the authors in NumPy, not a third-party one, agrees on the certification round and class of $68/68$ sampled roots; and the readout margin test is confirmed by arbitrary-precision arithmetic. Three of 1.16×10^8 FP32 prefix bits are pinned to the reference rather than proved, each one float32 ulp from θ (FP64: zero). On this model the cost is 48–$220\times$ this paper's own plain forward on the same device at matched accuracy, counted against that forward; the 362–$553\times$ of Table 5 are other models and other comparators, named in its caption.

I.4 The certificate on a published-configuration SHD trunk

This study runs the same certificate on the strongest SHD configuration in the paper, and on one this paper did not design. The trunk is *the published configuration*: the DCLS-delays architecture of Wang et al. (2024) at 2×256 neurons, $T = 100$, $\beta = 1/1.005$, reset-to-zero, 325,672 parameters, trained by the authors in one run of 150 epochs to reproduce the published number. Its own published readout scores 0.939929 ($2{,}128/2{,}264$) on this checkpoint, against the paper's reported $95.07 \pm 0.24\%$, which is a mean over seeds; one seed of ours is 1.07 points short. The readout the certificate runs is *ours*: the published readout is a softmax over classes summed over time, which is not linear in the spikes, so the certificate's precondition fails on it as published. We therefore refit a $P = 4$ per-time-block linear head (1,024 features, $\alpha = 1$, centred ridge, chosen by 5-fold CV on the 8,156 training samples only), and the certified accuracy is *that* head's: 0.92182 ($2{,}087/2{,}264$), which is 1.81 points below the published readout on the same frozen trunk and 2.89 points below the paper's 95.07%. The per-block split is required rather than chosen: the plain time-summed head ($P = 1$, `lin_sum`) reaches 0.890459 and its best regularised setting 0.89973, neither clearing 90%. The $P = 4$ head's train accuracy is 0.996444 ($8{,}127/8{,}156$), so the 1.81-point cost is a generalisation

gap rather than underfitting. The certified accuracy is 92.18%; the 95.07% figure is not reproduced and is not carried anywhere in this paper.

Quantity	Value
Trunk	published DCLS configuration
Trunk width, horizon, reset	2×256, $T = 100$, reset-to-zero
Published readout, this checkpoint	0.939929 $(2{,}128/2{,}264)$
Certified head (ours, $P = 4$, $\alpha = 1$)	0.92182 $(2{,}087/2{,}264)$
Reference fp64 vs deployed fp32, class agreement	$2{,}264/2{,}264$
Reference fp64 vs deployed fp32, accuracy gap	0.0
Roots released	$2{,}007$ (88.65%)
Never released	257
Wrong releases, vs exact fp64 reference class	0
Wrong releases, vs deployed fp32 argmax	0
Wrong releases, vs SHD label	81 (model error, not a certificate error)
Released-class accuracy vs label	0.9596 $(1{,}926/2{,}007)$
Median certification round	1.0
Rounds executed	4
L1 / L2 enclosure violations, lower / upper	0 / 0
Coordinates audited	$547{,}127{,}296$
L2 fp32-vs-fp64 spike mismatches	10 of $71{,}868{,}416$
L1 fp32-vs-fp64 spike mismatches	1 of $64{,}913{,}408$

Table 35: The certificate at full scale on a published-configuration SHD trunk, official $2{,}264$-sample test split. The trunk is the published configuration and the readout is this paper's refit $P = 4$ linear head; the certified accuracy is 0.92182, not the published 95.07%. Coverage is $2{,}007$ of $2{,}264$, so 257 roots are never released. The one $t = 0$ prefix bit per round that the enclosure does not decide cannot make a release unsound, because rounds two to four release nothing new and the enclosure audit reports zero violations over $547{,}127{,}296$ coordinates. Timing in this study was taken on a shared machine with no exclusive block, so no timing claim is made from it.

The fp64 reference gate is re-verified after every change: the exact fp64 twin and the deployed fp32 trunk agree on the class of all $2{,}264$ roots with an accuracy gap of 0.0, and differ on 10 of $71{,}868{,}416$ layer-2 spike coordinates and 1 of $64{,}913{,}408$ layer-1 coordinates. The certificate releases $2{,}007$ roots and withholds 257; the proof closes at round 1 for every root it releases, and four rounds run in total. Of the $2{,}007$ released classes, 81 disagree with the SHD label, which is the refit head's own model error rather than a certificate error: against both the exact fp64 reference class and the deployed fp32 argmax the count of wrong releases is zero.

Data preprocessing, a stated deviation. The h5 `times` are in seconds and the published loader never rescales them, so the published `duration=config.time_step=10` integrates each whole utterance into *one* frame — verified on the box: `SpikingHeidelbergDigits(root,'frame',duration=10)` returns $(1, 700)$, which bins to a single timestep rather than the `time_step=10` the config advertises. The reading used here, for train and test alike, is the one self-consistent with the published `n_bins=5` $\rightarrow$ `n_inputs=140` *and* `time_step=10`: 1 ms frames, then a 10 ms pool per timestep (giving $T = 100$) and a 5-channel pool per input (giving 140), on a $(8{,}156/2{,}264,\ 100,\ 140)$ tensor. Both poolings are monotone sums of the same 1 ms counts, no event is dropped or duplicated, and every one of the 700 channels lands in exactly one input. An earlier 1 ms-timestep reading, which keeps only the first 100 ms of each utterance, trained to about 41% and was discarded.

What this study is not. It is one checkpoint (epoch 83 of one seed), one dataset, one leak, one head. It makes no timing claim: the rounds were wall-clock of an instrumented, unoptimised loop on a shared machine with no exclusive block taken, and the median round time is reported nowhere as a cost. The certified object is the fp64 branch, and its tie to the deployed fp32 path is a measured agreement rather than a proof. The sweep over the head's block count stops at $P = 31$, so $P = 62$ and $P = 124$ are not measured .

I.5 What the fused kernel's gain consists of

The serving comparison of §5.5 runs the fused kernel against a step-wise LIF that pays one host dispatch per timestep. Three controls separate the kernel from the launcher. First, a cuda graph of the *same* step-wise loop, which replays it with one submission: over 60 paired, interleaved rounds it is faster than the fused kernel at the cells this paper reports ($32\times24\times4$: $0.59\times$, 47 of 60; $32\times24\times64$: $0.64\times$; $128\times24\times64$: $0.63\times$), and the fused kernel leads only once the device is loaded ($512\times24\times256$: $1.54\times$; $512\times250\times64$: $4.13\times$). Second, in kernel-only time the fused kernel is $40\times$ cheaper than one dispatch per timestep at $32\times24\times64$ but $2.1\times$ at $512\times250\times256$: the advantage *shrinks* with scale, which is the signature of launch elimination rather than of parallelism. At the reported operating point the kernel occupies a single block on one of 24 SMs, 0.09% of thread capacity. End to end, on the serving loop of §5.5, that same kernel gives $40\times$ less kernel time at the paper's cell, in 4 launches per window against 169, and its largest edge is where the device is loaded ($4.13\times$ at $512\times250\times64$, against $2.1\times$ at $512\times250\times256$). Third, against independently authored runtimes at matched accuracy the margin is 1.1–$1.8\times$ (Appendix I.9). The gain is therefore one submission per window instead of one per timestep, and it is reported as such.

I.6 Which recurrence the certificate covers

Theorem 3 is stated for Eq. (1), whose reset is *subtractive*: $x_t = u_t - \theta s_t$. Popular implementations of the same neuron reset to zero instead, $x_t = (1-s_t)u_t$. On the 512-neuron model of Appendix I.3, with identical currents, β and θ, the two conventions disagree on 4,094,604 of 115,916,800 spike coordinates (3.53%) and on 318 of 2,264 classes (14.0%), at task accuracies 0.7072 and 0.7694. Certifying Eq. (1) on a reset-to-zero network certifies a different network.

The proof of Theorem 3 uses the sign of the reset coefficient: nonpositive coefficients make Φ order-reversing, which is what produces the sandwich. Subtract-θ has that sign. Reset-to-zero has it only while the membrane is nonnegative, and here it is not: 81.6% of this model's membrane coordinates are negative, where the reset coefficient is $-u_t > 0$. Applied literally, the box-corner enclosure of Appendix G.8 therefore violates $\underline{z} \le z^* \le \overline{z}$ on that model 445,688 coordinate-rounds, i.e. summed over the 100 rounds and not per round, with margins up to 11.26. The same machinery on the subtract-θ recurrence violates it 704 coordinate-rounds over the same rounds, a factor 633 fewer, and each of those sits one float32 ulp from θ — near-ties rather than a structural failure.

The enclosure used for the 512-neuron result is therefore re-derived without the sign assumption. It is a proved sound interval extension, not a heuristic.

Proposition 11 (Sound interval extension of the reset-to-zero step)**.** *Let $\beta \ge 0$, $\theta \ge 0$, and let $[x_L, x_U] \ni x_{t-1}$ and $[I_L, I_U] \ni I_t$ be sound. Put $u_L = \beta x_L + I_L$ and $u_U = \beta x_U + I_U$, and define*

$$E([x_L, x_U], [I_L, I_U]) = \begin{cases} [0, 0], & u_L \ge \theta, \\ [u_L, u_U], & u_U < \theta, \\ [\min(u_L, 0),\ u_U], & \textit{otherwise.} \end{cases} \tag{7}$$

Then $x_t \in E$ for every admissible x_{t-1}, I_t, and E is inclusion-monotone: $[x_L, x_U] \subseteq [x'_L, x'_U]$ together with $[I_L, I_U] \subseteq [I'_L, I'_U]$ implies $E \subseteq E'$.

Proof. $\beta \ge 0$ gives $u_t = \beta x_{t-1} + I_t \in [u_L, u_U]$. If $u_L \ge \theta$ then $u_t \ge \theta$, so $s_t = 1$ and $x_t = (1 - s_t)u_t = 0$. If $u_U < \theta$ then $u_t < \theta$, so $s_t = 0$ and $x_t = u_t \in [u_L, u_U]$. In the remaining case either $u_t < \theta$, where $s_t = 0$ and $x_t = u_t \in [u_L, u_U]$, or $u_t \ge \theta$, where $x_t = 0$; hence $x_t \in [\min(u_L, 0), \max(u_U, 0)]$, and $\max(u_U, 0) = u_U$ because $u_U \ge \theta \ge 0$. For inclusion-monotonicity, u_L and u_U are nondecreasing in the endpoints since $\beta \ge 0$, and enlarging the inputs can only move a coordinate from the second case to the third to the first, each of whose expressions is monotone in u_L, u_U. □

Corollary 12 (What replaces order reversal)**.** *Let $\hat{\Phi}$ apply (7) coordinatewise, $(\hat{\Phi}([l, h]))_t = E([l_{t-1}, h_{t-1}], I_t)$. Then $\hat{\Phi}$ is inclusion-monotone on interval traces; every iterate that starts from a sound enclosure and is intersected with its predecessor stays sound; and a coordinate may be released exactly when its two endpoints agree.*

Proof. Inclusion-monotonicity is Proposition 11 coordinatewise. Soundness of each iterate follows by induction on t from soundness of the initial enclosure, since $\hat{\Phi}$ maps sound enclosures to sound enclosures. If the endpoints agree at t the iterate still contains x_t^*, so that coordinate is pinned. □

This is what stands in for Theorem 3's order reversal when the reset coefficient can be positive: soundness is carried by inclusion-monotonicity of an interval extension rather than by antitonicity of Φ, and the sandwich is replaced by an intersection of successive sound enclosures. On the subtract-θ recurrence (7) reduces to the shipped box-corner rule, so it strictly contains that rule rather than replacing it. The 704 residual violations of the box-corner chain reported above are therefore floating-point boundary effects one float32 ulp from θ, where the two readings of H meet, and not an enclosure failure: the margin test withholds a class at such a tie rather than misreleasing one.

Two things are still not claimed. The rule is sound but not tight — at $u = \theta$ the cases overlap and it may return a width where the reference is a point, so a coordinate can fail to certify. And the theorems of §4 cannot simply be restated for reset-to-zero. The sandwich of Theorem 3 fails there, and it fails by publishing a wrong coordinate with a zero error term rather than by being loose. The minimal instance is three steps: $T = 3$, $\beta = 1/2$, $\theta = 1$, $x_{\text{init}} = 0$, $I = (3, -\frac{1}{2}, 1)$, whose iterates are $s^0 = 000$, $s^1 = 111$, $s^2 = 101$ and $s^3 = s^* = 100$. At $K = 2$ the disagreement set is $C_2 = \{1\}$, so $U_2 = C_2 \cap [2, 3) = \varnothing$ and the box the sandwich would publish is the *point* 101 while the reference is 100; the readout bound $|M|\mathbf{1}_{U_K}$ is zero exactly where the bit is wrong. The failure is structural rather than incidental. The reset-to-zero map is order-reversing only while every admissible membrane is nonnegative, which is the condition this model violates; and at budget K the sandwich mechanism releases the K prefix coordinates and nothing more, whereas the subtract-θ recurrence releases those plus every agreement beyond K, all proved. The wider enclosure of Proposition 11 is therefore forced rather than convenient: it carries soundness by inclusion-monotonicity, which needs no sign assumption, instead of by antitonicity of Φ. A general failure template for the port and an exhaustive enumeration over 25,170 small systems (no counterexample for $T \leq 2$; 1.52% of systems fail from $T = 3$ on) are recorded in the evidence ledger as the reset-to-zero tightness study.

I.7 The certificate on the learned compound graph

Inside the learned compound graph the certificate runs on F's trained LIF layer and releases every root with zero wrong release, under the same exact-dyadic conditions and the same audit used elsewhere in this appendix. The same *margin test* also runs on the 512-neuron 76.9% model of Table 16, whose leak $\beta = 19/20$ leaves the dyadic conditions; the *enclosure* there is not the same one. Theorem 4 needs only sound Boolean bounds, so its test is unchanged, but reset-to-zero lies outside Theorem 3: that study takes its bounds from the sound interval extension of Proposition 11 with Corollary 12, not from the subtract-θ sandwich. There the reset-to-zero enclosure releases all 2,264 classes with zero wrong at median round 11 of 100 (Appendix I.3), where the hull rule reaches all 2,264 at a median round of 12 on the same recurrence. No matched-kernel prototype comparison was run on these dependent graphs: the same-kernel control of this paper is ELYSIA-P, its own stripped prototype, and it is measured only on the independent-task streams above.

I.8 Reuse on a 4,800-node dependent graph

A larger revision study runs the same question on a three-level multi-parent hypergraph over the full SHD test split with the 512-neuron model of Appendix I.3: 64 roots of 64 sources, 8 groups, 2 heads and one root, 4,800 nodes, on an RTX PRO 6000 Blackwell. A revision rebinds one source's sample and invalidates 4–8 nodes against 75–150 under recompute-from-scratch. Every arm is compared against a float64 recomputation the authors wrote, with a different summation order: 8,588,800 root-logit comparisons, zero class disagreements, and the checkpoint's own accuracy 1,742/2,264 reproduced exactly.

Per revision the system beats recompute-from-scratch by 1.21–1.24× over nine repeats at all four densities, losing in no cell. Against the authors' own hand-written memo and dependent-walk the ordering reverses: the system is 1.06–1.08× slower per revision in every cell (0/36 repeat-level comparisons, non-overlapping ranges), and what that gap costs is the artifact contract's bookkeeping, 0.246–0.250 ms per revision against 0.010–0.018 ms for the hand-written walk. End to end, the

per-revision advantage survives at revision density $d = 1$ (0.965) and ties at $1/2$ (0.999), but a per-completion cost of 110–169 ms against 78–80 ms for every other arm makes it lose at $1/8$ (0.838) and $1/32$ (0.753). Completion medians are noisy under host preemption, so the interval ratios are approximate while the per-revision numbers are stable. No matched-kernel prototype comparison on the learned dependent graphs is measured here.

I.9 Independently authored runtimes at matched accuracy

The serving comparison of §5.5 runs the fused kernel against a step-wise path the authors wrote. This appendix adds runtimes the authors did not: `snnTorch` 1.0.0 and `SpikingJelly` 0.0.0.0.14, the library the paper's own baseline already uses. All paths share one timing boundary — the batch already resident on the device as $x[B, T, 700]$, through the transposed projection, all $T = 100$ steps, temporal pooling and the readout, ending when $\mathrm{logits}[B, 20]$ exist — and one weight tensor, the 512-neuron model of Appendix I.3.

Matched accuracy is exact rather than approximate: every path, including the paper's own from-scratch reference run unmodified, returns $1{,}742/2{,}264 = 76.9435\%$ at batch 128, 256 and 2,264. There is no accuracy gap to declare.

Inference path	Device ms	vs fused	Steelman ms
Fused kernel (this paper)	**45.800**	—	**45.800**
SpikingJelly `LIFNode`, multi-step	52.188	1.14×	51.769
This paper's own PyTorch LIF	53.859	1.18×	52.987
`snnTorch` `Leaky`, projection hoisted	74.228	1.62×	60.403
`snnTorch` `Leaky`, canonical per-step	83.329	1.82×	70.171

Table 36: One forward over all 2,264 test samples, batch 2,264, CUDA events, paths interleaved round-robin inside one lock so that thermal drift affects every arm alike (a drift control measured 1.00–1.04×). The steelman column gives every path but the fused kernel — including this paper's own PyTorch LIF — a full cuda-graph replay, removing its Python dispatch entirely, while the fused kernel stays plain eager: it still leads by 1.13–1.53× over the two independently authored runtimes.

Two caveats bound what this shows. The margin over the paper's own pure-PyTorch reference is 1.18×, so most of the distance to the external libraries is their loop dispatch rather than fused-kernel arithmetic, consistent with Appendix I.5. And batch-1 figures are deliberately not reported: with a per-request boundary the timing window spans the host preparing 100 launches, so those numbers measure Python dispatch, not the device.

A stream defect this measurement found. `cuda_lif.py` launches its kernel with no stream argument, so every launch targets the legacy default stream and the launch error is never checked. Under cuda-graph capture the current stream is a private capture stream, so the kernel is not recorded and no error is raised. The numeric signature is unambiguous: replaying a captured graph returns accuracy $103/2{,}264$, and 103 is exactly the number of test samples labelled class 0, which is what constant logits with a fixed argmax predict. Two consequences follow. The kernel cannot benefit from the launch-overhead removal that both external runtimes can, so the steelman column above is biased *against* them; and in any multi-stream deployment the kernel runs on the wrong stream silently. The fix is to pass `at::cuda::getCurrentCUDAStream()` and to check the launch error.