AN EXECUTION CONTRACT FOR LAW-PRESERVING ROW EVICTION IN VERIFIER-GUIDED SEQUENTIAL MONTE CARLO

Anonymous authors
Paper under double-blind review

ABSTRACT

A language model generating candidates in a batch may keep computing after they finish or can no longer succeed. Verifier-guided sequential Monte Carlo (SMC) weights candidates using external checks. We specify how to remove unnecessary model rows while preserving selection weights, score updates, and next-token probabilities. Under this execution contract, we prove that row removal preserves the output distribution and test agreement under shared randomness. A deterministic reference produces identical candidates with positive weight and identical selected outputs across four execution policies on 219 of 256 Python tasks. None of the policies retains a candidate with positive weight on the other 37 tasks. Across four benchmarks, removing finished rows saves 32.4 to 45.4% of model-row evaluations. Removing candidates whose zero weight cannot be repaired adds statistically resolved savings on arithmetic tasks, including fresh problems and harder variants. Additional savings remain unresolved on code and database queries. Throughput comparisons are descriptive because the policies reach different peak numbers of resident rows.[1]

1 INTRODUCTION

Consider a language model that writes several candidate programs in one batch. One finishes early. Another raises an irreparable exception. Neither needs further model computation, but the finished program may still be selected. We ask how to remove their rows while preserving output probabilities.

In *verifier-guided generation*, an external check supplies feedback while we generate tokens, a process called *decoding*. Sequential Monte Carlo (SMC) maintains N weighted candidates, called *particles*. Checks supply nonnegative scores, called *potentials*, that multiply a particle's weight. At an update point, or *checkpoint*, we apply accumulated scores and may *resample*: we draw a new population in proportion to the weights, allowing copies of a candidate. At termination, we select one candidate in proportion to its final weight. Positive weight means eligibility under our generation-time checks. It does not establish correctness on hidden evaluation tests.

We separate two records of each candidate. Its *logical slot* retains the text, weight, ancestry, update clock, and random stream needed for sampling. Its *physical row* holds the model tensors needed to extend it. The slots form the logical population and the rows form the physical batch.

Our verifier scores *semantic units*, such as complete top-level code blocks or calculator annotations. Later tokens cannot revoke a completed unit's boundary. A zero score is *absorbing* if no continuation can repair it. We call it a *certified zero* when the verifier establishes this property. The *unit-barrier* (UB) schedule waits until every live particle reaches the next unit level, a shared checkpoint called a *barrier*. The *eager-prune barrier* (EPB) schedule acts on certified zeros immediately and keeps positive updates at that barrier. *Zero compaction* removes zero-weight rows, including finished candidates that fail a visible test. *Termination eviction* removes every finished row and retains its selection state as *parked state*. EPB with zero compaction is *compact EPB* (cEPB). We append "+evict" when we also use termination eviction. Figure 1(b) to (d) illustrates all four policies with scripted tokens. A *row-forward* evaluates one resident model row for one token step.

[1] Artifact: https://anonymous.4open.science/r/ICLR-SMC.

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.

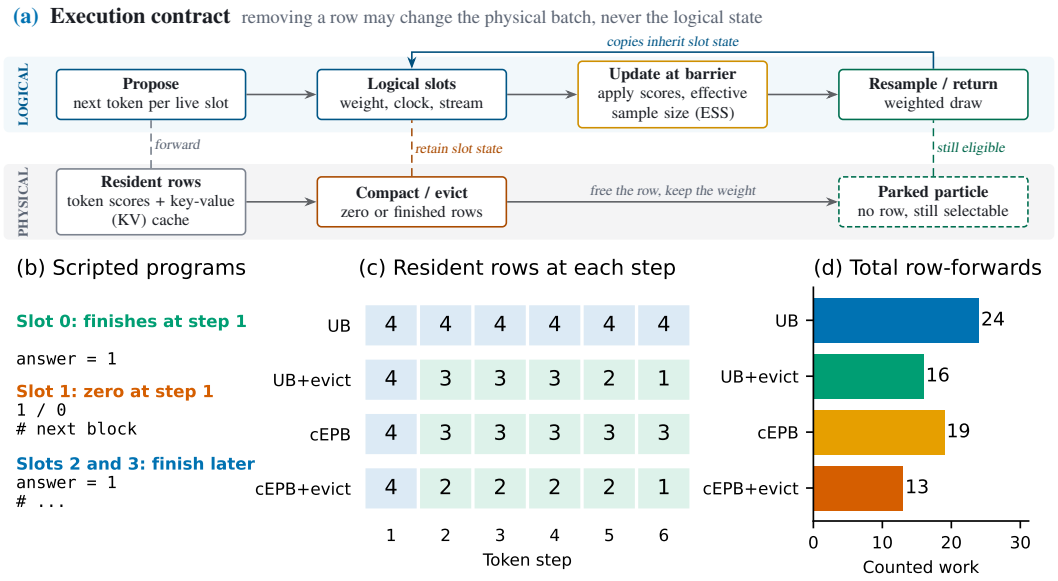


Figure 1: **Execution contract and a deterministic illustration on a central processing unit (CPU).** (a) Logical records determine selection. Physical rows hold token scores and key-value (KV) attention state. Effective sample size (ESS) measures weight concentration. (b) Four particles use scripted token fragments and the Python verifier. Slot 1 executes a completed block that raises `ZeroDivisionError`. Slot 0 finishes with positive weight at step 1, and slots 2 and 3 finish at steps 5 and 6. (c) Resident rows per step under unit barrier (UB), compact eager-prune barrier (cEPB), and optional finished-row eviction (+evict). (d) Row-forwards count one evaluation of a resident row at one token step. All arms advance the barrier once per ready checkpoint and return the same program, checked against an independent finite reference. This is an executable illustration, not a model-generated benchmark or a timing measurement.

Semantic SMC combines language-model proposals with external potentials (Loula et al., 2025). Our *execution contract* specifies what physical row removal must preserve. The *proposal kernel*, which supplies next-token probabilities, must stay fixed when the batch changes. We also retain a shared barrier clock and, for our comparison under shared randomness, streams indexed by logical slot. Under these conditions, UB and EPB have the same population distribution at fixed N (Proposition 1). We distinguish *law equality*, meaning equal output distributions, from *coupled identity*, meaning identical paths or draws under shared randomness. Accelerator routines can depend on batch shape, so we test this numerical behavior separately.

We test the contract on exact targets and accelerator workloads, then compare the work and throughput effects of finished-row removal and early pruning.

2 Target Distribution and Verifier Contract

We first specify the output distribution that row removal must preserve. Let T be the token limit. At token position $t \in \{1, \dots, T\}$, let $x_{<t}$ be the generated prefix, v a candidate next token, and ℓ the language-model (LM) logit vector, the scores before normalization. The softmax function converts these scores to probabilities. With sampling temperature $\tau > 0$, our next-token proposal is $q_\tau(v \mid x_{<t}) = \mathrm{softmax}(\ell/\tau)_v$. We use nucleus threshold $p = 1$, retaining the full probability distribution instead of restricting sampling to its highest-probability tokens. A sequence terminates at an end token, a configured stop string, or the token limit T. At the limit we evaluate the truncated text using the same terminal constraint. We score its completed units and pass any incomplete remainder only to the terminal check. For a complete sequence x, let $q_\tau(x)$ be the product of its token probabilities, $M(x)$ its unit count, and $u_{1:m}$ its first m semantic units, where m ranges from 1 to $M(x)$. The terminal factor $H(x) \in \{0, 1\}$ equals one exactly when x satisfies the declared generation-time requirement. The score $g_m(u_{1:m}) \geq 0$ evaluates unit m given the preceding units.

We write π_τ for the target distribution and Z_τ for the constant that normalizes its probabilities:

$$\pi_\tau(x) = Z_\tau^{-1}\, q_\tau(x)\, H(x) \prod_{m=1}^{M(x)} g_m(u_{1:m}), \qquad Z_\tau = \textstyle\sum_x q_\tau(x) H(x) \prod_m g_m(u_{1:m}), \tag{1}$$

The sum ranges over complete sequences. Each output receives probability according to its model probability and verifier scores if it passes the terminal check. We use q_τ both to generate candidates and as the target's base distribution. Appendix E gives the correction for a different base. Positive scores express preferences among outputs that satisfy H, so validity alone does not determine our target. Our *uniform-valid-of-N* baseline selects uniformly among valid members of N independent samples. We measure agreement with the target using *total variation* (TV) distance, the largest probability difference that two distributions assign to the same event. On our exactly enumerable diagnostic, TV is 0.556 for uniform-valid-of-N and 0.021 for UB at $N = 32$ (Table 9). We therefore evaluate *posterior fidelity*, agreement with the target, separately from task pass rate.

Our verifier must satisfy four properties to preserve this target.

An early zero must be irreparable. We remove a zero-scored prefix only if every append-only completion has terminal weight zero. Such removal is *support preserving*: it discards no output with positive target probability. A later Python function definition can repair a failed test, so we run visible tests only at termination. A trailing WINDOW clause can likewise repair a Structured Query Language (SQL) prefix. Our SQL verifier certifies only failures that no suffix can repair, such as an unknown table or function.

An inconclusive check must remain neutral. Unsupported segmentations, statically rejected constructs, and timeouts return the multiplicative factor 1. Failure to check does not establish invalidity. An uncaught exception in a completed top-level block, however, is absorbing because later blocks cannot undo it.

The verifier must be deterministic and pure. Its result must depend only on the prefix. We therefore reject file, process, network, and nondeterministic imports in generated code. A wall-clock limit still lets host load determine whether a check finishes or abstains. Runs with such limits satisfy this premise only when no timeout fires. Table 4 reports timeout counts, and Table 3 lists the premises.

Generation feedback must be separate from evaluation. We distinguish scores used to guide generation from tests used only to evaluate the returned output. Mixing these roles can discard outputs for reasons outside the declared target.

3 Execution Contract and Guarantees

We now turn the target into an execution rule. Removing a row should change model work while preserving the logical state that determines selection. We distinguish *consistency*, convergence to the target as $N \to \infty$, from *finite-N equivalence*, equality between population distributions at a fixed number of particles.

3.1 Completed Units and Exactly-Once Application

Our *segmenter* identifies semantic units. It is *append-stable* if later tokens never revoke an emitted boundary. At a checkpoint, we apply the scores received since the previous checkpoint. We maintain their cumulative product and divide by the product already applied, so each factor enters the weight exactly once. Unsupported spans contribute 1.

3.2 Schedule, Zero Compaction, and Termination Eviction

Table 1 separates update timing from row removal. UB applies all scores at the common unit barrier. EPB acts on absorbing zeros immediately but keeps positive updates at that barrier. cEPB also removes zero-weight rows, including rows whose terminal factor sets their weight to zero. Termination eviction removes every finished row while parking its positive selection weight. Most rows that cEPB removes on code are finished programs that fail their visible test (§5.3). To decide when to resample, we measure weight concentration with the *effective sample size* (ESS). For particle

Table 1: The design space separates the logical update schedule (top) from the physical row-removal policy (bottom). A configuration combines one choice from each panel: compact eager-prune barrier with eviction (cEPB+evict) uses EPB updates and removes certified-zero rows and every terminated row.

A. Logical update schedule

Schedule	Observe	Apply zero	Apply positive	Guarantee
Unit barrier (UB)	completed unit	common barrier	common barrier	same law as EPB
Eager-prune barrier (EPB)	completed unit	immediately	common barrier	same law as UB

B. Physical row-removal policy

Policy	Certified zero	Terminated row	Logical state retained
Dense	no	no	all slots
Zero compaction ("compact")	yes	zero weight only	positive slots
Termination eviction ("+evict")	no	yes, any weight	parked weight and final state
Both	yes	yes, any weight	parked positive slots only

Finite-N equivalence requires the unit-step barrier clock of Algorithm 1, fixed proposal kernels, deterministic verification, and slot-indexed random streams.

index $i \in \{1, \dots, N\}$, let w_i be its unnormalized weight and $\bar{w}_i = w_i / \sum_j w_j$ its normalized weight, with j ranging over all slots. Then ESS is $1/\sum_i \bar{w}_i^2$. Every implemented schedule resamples when ESS $< N/2$.

3.3 Physical compaction and deterministic reuse

A *tombstone* records a particle that can no longer be selected. Marking it this way saves work only if we also stop evaluating its row. We compact the batch by applying the same survivor indices to logits, particle metadata, and every layer's *key-value (KV) cache*, which stores attention information from earlier tokens (Algorithm 1). We also reuse verifier results stored by prefix, a technique called *memoization*, and copy interpreter state from that prefix. Under our deterministic-verifier contract, this reuse changes execution counts without changing verdicts. Dense execution still forwards resident finished rows, whose outputs do not extend the logical text. Eviction removes those rows. At resampling, dense execution rebuilds both live and finished descendants, while eviction rebuilds only live descendants. At termination, including the token limit, we apply pending completed-unit factors and the terminal constraint once. If every weight is zero, we report *collapse*. The implementation returns fallback text, which counts as failure.

3.4 What is guaranteed

We use *multinomial resampling*: we independently draw N ancestors from the normalized weights. Its conditional resampling error *vanishes* as N grows if averages of every bounded test function over the new population converge in probability to the weighted averages before resampling. Appendix E states this condition formally.

Proposition 1 (Consistency and finite-N equivalence)**.** *Assume a finite horizon, a uniformly bounded number of units and bounded unit scores, append-irreparable zeros, append-stable segmentation, a pure deterministic verifier and a positive target normalizer. For (ii), additionally assume a fixed proposal kernel, meaning a next-token distribution independent of the physical batch, read through slot-indexed streams on the unit-step barrier clock of Algorithm 1. (i) With vanishing conditional resampling error, the implemented multinomial sampler is consistent for the target of Equation 1 at termination. (ii) At every barrier stopping time (the first token step at which the barrier is ready), the positive-weight particles, their texts and normalized weights, have the same joint law under UB and EPB, with or without physical compaction and eviction of terminated rows, and so does the terminal draw. Zero-weight rows, which no later operation reads, may differ.*

Sketch. For (i), whether a barrier is ready and ESS is below $N/2$ depends only on the observed history. These update times are therefore *stopping times*, whose occurrence requires no knowledge of future tokens. A conditional law-of-large-numbers induction gives consistency (Del Moral et al., 2012, Whiteley et al., 2016). For (ii), we run the algorithms with shared randomness. An absorbing-zero prefix can never be selected at a later checkpoint. A finished particle keeps the same parked weight

Algorithm 1: Unit barrier (UB) and compact eager-prune barrier (cEPB) with optional termination eviction. Logical slots retain selection state after their model tensors are removed. Only the zero branches differ. Compaction selects logits, metadata, and every key-value (KV) cache layer with the same survivor indices. The flag e enables eviction when $e = 1$. The flag f_i records whether slot i has been finalized.

Input: proposal distribution q_τ, prefix verifier V, terminal factor H, particle width N, token horizon T, schedule $s \in \{\mathrm{UB}, \mathrm{cEPB}\}$, and termination-eviction flag $e \in \{0, 1\}$

Data: logical slot i: text x_i, log weight $\log w_i$, log cumulative potential already applied $\log G_i$, completed-unit count M_i, finalization flag f_i, and state in $\{$LIVE, PARKED, ZERO$\}$, barrier level b, and token step t

Initialize N LIVE slots with $\log w_i = \log G_i = M_i = f_i = b = t = 0$
while *a slot is* LIVE *and token step* $t < T$ **do**
  Sample for each LIVE slot from its slot-indexed stream, increment t, and mark end tokens, stop strings, and $t = T$ as termination
  Forward every resident row and add its count to row work. Ignore outputs for finished logical slots and update completed-unit counts M_i
  foreach LIVE *slot* i *with a new completed unit* **do**
    if $s = \mathrm{cEPB}$ *and* $V(x_i) = 0$ **then** finalize i as ZERO, set $\log w_i = -\infty$, and remove its physical row
  foreach *slot* i *that terminates* **do**
    if $s = \mathrm{cEPB}$ *and its terminal update, applied now, gives* $\log w_i = -\infty$ **then** finalize i as ZERO and remove its physical row, else mark i PARKED, retain $(x_i, \log w_i, \log G_i, M_i, f_i)$, and remove its physical row if $e = 1$
  if *every* LIVE *slot has* $M_i \geq b + 1$*, or no slot is* LIVE **then**
    $b \leftarrow b + 1$ exactly once, independent of removal and resampling
    apply each unfinalized slot's new potential ratio and apply $H(x_i)$ once to newly terminated slots, then finalize any resulting zero
    if *all weights are zero* **then** report collapse and **return** failure
    if $\mathrm{ESS} < N/2$ *and a slot is* LIVE **then**
      resample N logical slots from LIVE and PARKED weights, copy finalization state, renumber streams by offspring position, reset weights, and rebuild resident rows for live offspring and, if $e = 0$, parked offspring
Terminate any remaining live slot. Apply pending completed-unit factors and H once to every unfinalized slot, park positive slots, and report collapse if all weights are zero. **return** a normalized-weight draw from parked slots

as in the dense runner. The barrier level b advances by $b \leftarrow b + 1$ once per barrier, independently of removal and resampling. We reassign proposal streams only by offspring position after resampling. Readiness, ancestor probabilities, ESS decisions, and terminal draws therefore agree. Appendix E gives the formal induction and observed-history construction, called a *filtration*.

For example, suppose a finished candidate has weight 1 and a second later finishes with weight 3, without intervening resampling. If these are the only candidates with positive weight, the first must still be returned with probability $1/4$. Keeping its text but dropping its weight makes that probability zero. We therefore retain its selection weight when freeing the model row.

Scope of the result. Part (i) uses standard adaptive-SMC consistency. Part (ii) makes row removal testable. Slot-indexed streams suffice for our coupling proof, but are not universally necessary for marginal-law equality. Moving positive updates and resampling times retains (i) but loses (ii). Part (ii) extends to systematic and stratified resampling, which coordinate ancestor draws, when runs share ordered slots and resampling variables. We claim (i) only for our multinomial implementation.

4 Experimental protocol

Our experiments follow the contract's obligations. We first test sampling behavior, then measure work savings and elapsed time. Table 3 states which assumptions each study satisfies. Appendix A gives the protocol details.

Contract audits. We use a target with nine outputs of positive weight and calculate their probabilities exactly. It includes positive scores, absorbing and terminal zeros, and unequal completion lengths. A separate trace delays a zero-scored unit to test the barrier clock. We run 20,000 trials per

Table 2: Exact-target contract audit (20,000 central processing unit (CPU) trials per cell). The reference pair uses the unit-step clock, and each later row changes one contract clause. TV is total variation distance from the exact target, with subscripts giving particle counts 16 and 64. The target normalizer is Z, and Rel. Z bias is the relative error in its estimate. Δ work is the change in proposed logical tokens relative to compact eager-prune barrier (cEPB). CI means confidence interval. Diff. counts unequal returned draws in 2,000 trials with shared randomness.

Configuration	TV_{16}	TV_{64} [95% CI]	Rel. Z bias	Δ work	Diff.	Exposes
UB = cEPB (reference)	.049	.015 [.010,.022]	−.001	baseline	0	none
Lag-bounded positive updates	.051	.018 [.012,.025]	−.001	+1%	975	schedule change
Discard a parked weight	.127	.120 [.113,.127]	.302	+6%	1,253	law, coupling
Prune nonzero-weight rows	.309	.309 [.309,.309]	−.312	−18%	887	law, coupling
Key store by unit count	.972	1.000 [1.000,1.000]	−1.000	−53%	2,000	law, coupling
Inherit parent random stream	.044	.015 [.010,.022]	−.002	0%	1,118	coupling
Restart after collapse	.049	.015 [.009,.022]	−.001	0%	0	not triggered

Notes. Work and coupled differences use $N = 64$. The lag-bounded row allows positive updates before the barrier and is a valid alternative schedule, with no defect exposed. Collapse occurs only at $N = 2$, where the restart mutation changes 132 of 2,000 paired draws. Appendix E gives the full audit and the slow-zero trace.

Table 3: Premises of Proposition 1(ii) met by each evidence tier. Fixed kernel means batch-independent token probabilities. Pure verifier means a deterministic prefix-only result. Slot RNG means a random-number generator indexed by logical slot. Clock means the shared unit-step barrier. ✓*: Python and SQL runs are pure only when no wall-clock limit fires (timeout counts in Table 4). The arithmetic verifier has no timeout.

Evidence	Fixed kernel	Pure verifier	Slot RNG	Clock	Establishes
Exact target, mutants (Table 2)	✓	✓	✓	✓	exact-target diagnostics
GPU, invariant kernels (§5.2)	✓	✓*	✓	✓	full-cohort identity
GPU, default kernels (§5.2)	no	✓*	✓	✓	coupling differences
Matched cohorts (Table 4)	no	✓*	no	✓	rows, zeros, quality
Shared batch (§5.4)	no	✓*	no	✓	descriptive throughput
Historical scaling	no	✓*	no	no	before clock repair

cell at $N \in \{2, 4, 16, 64\}$, changing one contract requirement at a time (Table 2). On the accelerator, we rerun the complete Mostly Basic Programming Problems (MBPP) cohort with slot-indexed streams under default kernels and under deterministic kernels whose results are independent of batch composition (§5.2).

Matched cohorts. Our four experimental configurations, or *arms*, are UB, UB+evict, cEPB, and cEPB+evict. We compare them on the Python benchmarks MBPP-sanitized (256 tasks) (Austin et al., 2021) and HumanEval (164) (Chen et al., 2021), BIRD Mini-Dev text-to-SQL (500) (Li et al., 2023), and Grade School Math 8K (GSM8K) arithmetic (200) (Cobbe et al., 2021). All arms use Qwen2.5-Coder-1.5B-Instruct (Qwen Team, 2024), $N = 8$, temperature $\tau = 1$, seeds $0, 1, 2$, one H100 accelerator, and the unit-step clock. We added GSM8K after inspecting the code and SQL results. We then froze a confirmatory protocol and predictions before rerunning the four arms on 300 fresh GSM8K problems and the corresponding GSM-Hard problems, which change only the numbers (Gao et al., 2023).

Contrasts and uncertainty. We define Δ_1 as UB+evict minus dense UB, isolating termination eviction, and Δ_2 as cEPB+evict minus UB+evict, isolating additional certified-zero removal. Each pair of arms shares a host, clock, model, verifier, and task grid. We pair observations by task and seed. After averaging seeds, we resample tasks 10,000 times to obtain bootstrap 95% confidence intervals (CIs). We also resample both tasks and seeds to check sensitivity to either source of variation. For quality tests and the two confirmatory Δ_2 tests, we use Holm's step-down correction to control the probability of any false rejection within the stated test family. An interval containing zero leaves a difference unresolved and does not establish equivalence. We run held-out EvalPlus 0.3.1 tests (Liu

Table 4: Primary same-host attribution with $N = 8$ particles (three seeds). Δ_1 is UB+evict minus dense UB and Δ_2 is cEPB+evict minus UB+evict, negative meaning less model work, with marginal 95% task-bootstrap confidence intervals (CIs) in brackets. Quality is held-out EvalPlus accuracy for code, execution accuracy (EX) for BIRD, and answer accuracy for GSM8K. Quality changes are in percentage points. UB is unit barrier, cEPB is compact eager-prune barrier, and +evict removes every finished row.

Benchmark	**UB rows**	**Eviction Δ_1**	**Early zero Δ_2**	**Zeros/task**	**Quality Δ_2**
MBPP-256	923	$-419\,[-447, -392]$	$-6\,[-19, 7]$	0.06	$+2.5\,[-0.6, 5.8]$
HumanEval-164	1669	$-719\,[-771, -666]$	$-13\,[-39, 13]$	0.20	$-1.6\,[-5.3, 2.0]$
BIRD Mini-Dev-500	718	$-232\,[-246, -219]$	$-4\,[-11, 2]$	0.38	$+0.7\,[-1.5, 2.9]$
GSM8K-200	1479	$-497\,[-537, -460]$	$\mathbf{-40}\,[\mathbf{-59}, \mathbf{-21}]$	1.53	$+2.5\,[-2.0, 6.8]$
Confirmatory protocol (predictions frozen before data collection, shared-batch runtime)					
GSM8K, fresh 300	1443	$-526\,[-564, -489]$	$-35\,[-49, -21]$	1.42	$-0.9\,[-4.2, 2.3]$
GSM-Hard, same 300	1751	$-550\,[-588, -513]$	$-113\,[-136, -90]$	4.36	$1.9\,[-0.8, 4.4]$

Notes. MBPP has $n_{\text{work}} = 256$ tasks and $n_{\text{hidden}} = 223$. HumanEval has 164 for both endpoints. Rows are physical model row-forwards per task. Zeros/task counts certified early prunes in cEPB+evict and excludes rows that terminate with zero weight. Neutral timeouts by arm (UB, UB+evict, cEPB, cEPB+evict): 21, 16, 6, 6 on HumanEval, none elsewhere (the purity condition of Table 3). Appendices A and B state the evaluation rules and zero-mass counts. No row carries a latency claim.

et al., 2023) only after fixing the returned draws. A run with no positive terminal weight counts as a failure.

Time. Our earlier isolated 30-task timing grids at $N \in \{32, 64, 128\}$ predate the clock repair. The work-attribution study uses a full H100 graphics processing unit (GPU). Our shared-batch study and step-time curve use a Multi-Instance GPU (MIG) 2g.20gb partition, with 20 GB of memory and two of seven compute slices. We call the number of resident rows the *occupancy*. A queued request enters when occupancy plus N fits an admission threshold $C \in \{64, 128\}$. Resampling can subsequently grow the batch beyond C, leaving unequal peak occupancy across arms (§5.4). Each request retains its unit-step clock and parked weights. However, its rows share a random generator consumed in physical row order, and its kernels see the pooled batch shape. This study therefore meets neither the slot-indexed stream nor the fixed-kernel premise.

5 Results: Contract Fidelity, Work, and Time

5.1 The Barrier Clock Is Part of the Algorithm

Agreement on the output can hide a scheduling defect. If a zero at unit 1 is removed while survivors have reached unit 2, deriving the barrier from their minimum unit count jumps the threshold to 2. UB and EPB then resample at different steps despite identical proposal kernels. Our unit-step rule advances the threshold once per ready checkpoint and never after removal or resampling. UB, UB+evict, EPB, cEPB, and cEPB+evict then agree on the slow-zero trace's clocks, weights, and ancestors (Figure 2(a)), as do 3,000 further paired runs on a central processing unit (CPU). Our mutation audit detects 4 of 5 violations at $N = 64$ and restarting after collapse at $N = 2$ (Table 2).

5.2 The Fixed-Kernel Premise Can Be Enforced on the Accelerator

The contract requires that changing the batch leave each surviving row's next-token probabilities unchanged. Our token-level probe finds violations under default 16-bit brain floating-point (bfloat16, or bf16) kernels. We locate batch-dependent calculations in matrix multiplication, root mean square normalization (RMSNorm), and attention. Fixing matrix tiling alone (He & Thinking Machines Lab, 2025) is insufficient. Controlling all three removes every logit difference in the probe (Appendix B). With these batch-independent operations, all four arms agree on 218 of 219 tasks where UB retains positive terminal weight. The exception also varies within one arm because the GPU adds token probabilities in a variable order, which can change a draw near a selection boundary. With a sequential scan, all four arms agree on 219 of 219 tasks, against 207 of 222 with default kernels (Figure 2(b)). Identity requires equal positive-weight populations and returned draws. By the draw alone, default

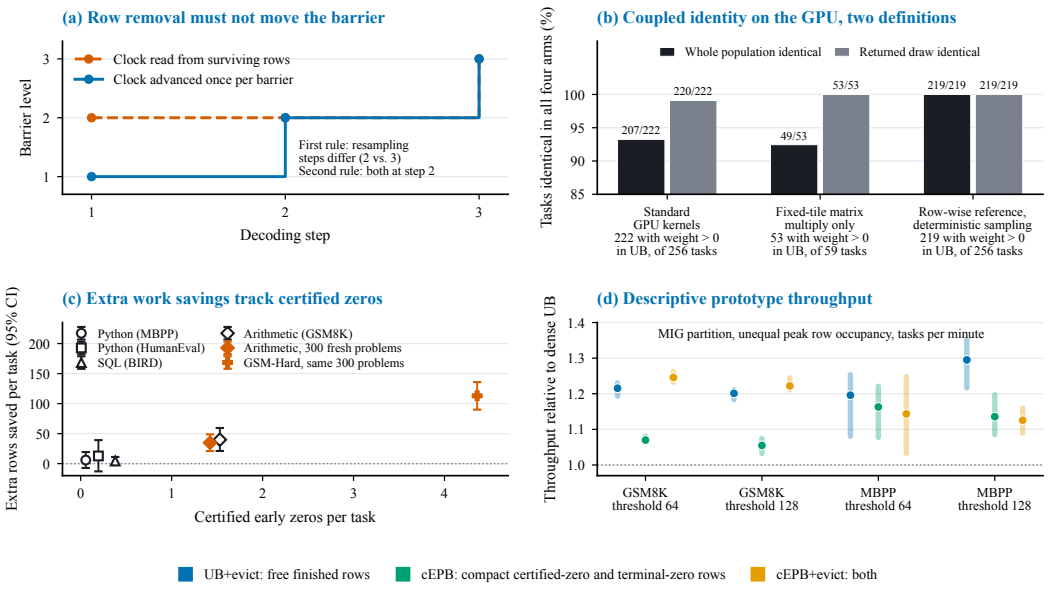


Figure 2: (a) Advancing the barrier once per ready checkpoint keeps unit-barrier (UB) and eager-prune barrier (EPB) resampling together. (b) Mostly Basic Programming Problems (MBPP) tasks with all four configurations identical, by positive-weight population and by returned draw, on tasks where dense UB retains positive weight. The matrix-only probe covers 59 tasks. Appendix C accounts for every task under both numerical modes. (c) Rows saved by adding certified-zero removal to finished-row eviction ($-\Delta_2$), red for the confirmatory cohorts (same 300 problems). (d) Tasks per minute relative to dense UB (seed means, three-seed ranges). Compact EPB (cEPB) also removes finished zero-weight rows.

arms agree on 220 of 222 tasks. These default-kernel differences measure agreement under shared randomness, not distance between output distributions. Neither complete cohort records a timeout. Full reference overhead at $N = 8$ or 64 remains unmeasured. Appendix B reports the partial timing probe, and Appendix C accounts for every task. Under the reference, all four arms collapse on the other 37 tasks. Under default kernels, 221 tasks have positive mass in every arm, 34 collapse in every arm, and one has positive weight in some arms but collapses in others. The reference also defines its own numerical proposal, which need not equal the default proposal.

5.3 Termination eviction and the role of certified zeros

Having tested sampling behavior, we now separate the two sources of saved work. Removing finished rows saves 45.4% of dense UB's row-forwards on MBPP and 43.1% on HumanEval (Table 4). After eviction, additional savings from certified zeros remain statistically unresolved on code and BIRD. They are resolved on GSM8K, where certificates occur more often.

We confirm the saving on 300 fresh GSM8K problems and their GSM-Hard variants. Larger operands increase certified zeros from 1.42 to 4.36 per task, with a paired difference of 2.94 $[2.53, 3.33]$. Both confirmatory contrasts have Holm-adjusted $p < 0.0001$. The cEPB arm without eviction also removes finished zero-weight rows. Appendix A examines runs without early pruning, without a complete breakdown by removal cause. Appendix C gives arithmetic certificate checks and a HumanEval timeout sensitivity analysis.

Held-out quality differences remain unresolved. For the six held-out Δ_2 contrasts in Table 4, the smallest Holm-adjusted p is 0.921. These are not equivalence results. By the planning counts in Appendix B, the four primary quality endpoints lack the planned power for a two-percentage-point pruning effect. This calculation does not cover the confirmatory cohorts.

5.4 SHARED-BATCH THROUGHPUT IN THE MEASURED PROTOTYPE

A smaller batch need not shorten a decode step, so we measure throughput separately (step-time curve in Appendix B). At admission threshold $C = 64$, UB+evict completes $1.22\times$ as many GSM8K tasks per minute as dense UB and $1.20\times$ as many MBPP tasks. Figure 2(d) summarizes these throughput comparisons.

Given eviction, pruning improves measured GSM8K throughput but reduces MBPP throughput. Appendix B reports both thresholds.

Other arms reach 76 and 142 peak rows while dense UB stays within 64 and 128 (Table 6). Without an equal hard capacity limit, these comparisons remain descriptive. The gap between the aggregate forward timer and a bare-step measurement suggests runtime overhead, but our instrumentation does not isolate its components or measure accelerator utilization (Appendix B). These measurements establish neither a latency benefit nor production throughput.

6 RELATED WORK

Verifier-guided generation. Semantic SMC (Loula et al., 2025) is our closest scheduling comparison. We study row removal under a fixed schedule. Learned *twists* steer proposals toward promising futures (Zhao et al., 2024, Kim et al., 2025), while adaptive weighted rejection sampling chooses candidate-token queries (Lipkin et al., 2025). Best-scoring selection (Sun et al., 2024) differs from our uniform-valid-of-N baseline, which chooses uniformly among valid samples. ChopChop monitors whether a prefix can still satisfy a constraint and could supply earlier zero certificates (Nagy et al., 2026).

Particle systems and inference runtimes. The Feynman-Kac framework of weighted paths supports our coupling (Del Moral, 2004, Chopin & Papaspiliopoulos, 2020). Memory-efficient SMC compacts ancestry (Jun & Bouchard-Côté, 2014), while asynchronous SMC removes our required barrier (Paige et al., 2014). Orca, PagedAttention, and SGLang reuse finished-request capacity (Yu et al., 2022, Kwon et al., 2023, Zheng et al., 2024). We must also preserve each freed particle's selection weight. Batch-invariant kernels enforce the fixed-kernel premise (He & Thinking Machines Lab, 2025). Appendix G gives further comparisons.

7 LIMITATIONS

Our guarantees require a finite horizon, bounded units and scores, and the stated verifier and kernel premises. Python and SQL purity require timeout-free runs. We evaluate one 1.5-billion-parameter model at $N = 8$ in a Hugging Face runtime. Our prototype lacks a hard row limit and paged KV memory, which allocates the attention cache in reusable blocks. Its MIG results do not establish production efficiency. Full reference overhead, asynchronous verification, and compute-limited full-GPU service remain unmeasured. Stronger models may change certificate frequency and timing, which accuracy alone cannot predict. Our simulation conditions on these opportunities. Held-out quality remains unresolved, and posterior fidelity is tested only on enumerable targets (Table 9, Appendix H).

8 CONCLUSION

A finished candidate no longer needs model computation, but it may still be selected as the answer. Our execution contract preserves this distinction when rows are removed. Across our evaluated workloads, removing finished rows accounts for most measured work savings. Early pruning adds savings when the verifier can establish failure soon enough, as in our arithmetic cohorts. These findings make termination eviction an important baseline for evaluating pruning. They also guide how we assess an implementation: check that removing rows preserves the sampling behavior, then measure whether the saved work improves throughput under an enforced resource limit.

Artificial intelligence (AI) use statement

We used generative AI tools to polish prose and assist with coding. We developed the research questions, method, experimental design, analysis, and conclusions. We reviewed and tested all AI-assisted code, and we regenerate numerical claims from released outputs. We take responsibility for the final manuscript and artifact.

Ethics and security statement

We use public benchmark data and involve no human subjects. Executing model-generated code creates security risk. Our artifact rejects file, process, network, and nondeterministic constructs and sets a time limit for each execution. On macOS, it also applies a Seatbelt sandbox profile that denies network access, process execution, writes outside a temporary directory, and reads from other user directories. Users on other platforms must run the worker inside an external operating-system or container sandbox. A fork alone is not a security boundary. The studies using NVIDIA's CUDA accelerator platform ran on Linux inside disposable GPU containers, where we checked generated code with an abstract syntax tree (AST) gate, a filter on its parsed structure, and enforced per-execution time limits. These containers did not use the Seatbelt profile.

Reproducibility statement

Appendix H specifies the contract-audit procedure. The accompanying artifact README gives validation and regeneration commands and maps reported quantities to retained measurements and protocols. Tables 10 and 13 use retained aggregate measurements for each configuration, stored in JavaScript Object Notation (JSON). Per-trial histories are not retained for these supporting simulations. Full-cohort quantities are regenerated from per-task rows and scores.

References

Lakshya A. Agrawal, Aditya Kanade, Navin Goyal, Shuvendu K. Lahiri, and Sriram K. Rajamani. Guiding language models of code with global context using monitors. *arXiv preprint arXiv:2306.10763*, 2023.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

David Brandfonbrener, Simon Henniger, Sibi Raja, Tarun Prasad, Chloe Loughridge, Federico Cassano, Sabrina Ruixin Hu, Jianang Yang, William E. Byrd, Robert Zinkov, and Nada Amin. VerMCTS: Synthesizing multi-step programs using a verifier, a large language model, and tree search. *arXiv preprint arXiv:2402.08147*, 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Nicolas Chopin and Omiros Papaspiliopoulos. *An Introduction to Sequential Monte Carlo*. Springer Series in Statistics. Springer, 2020.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Pierre Del Moral. *Feynman-Kac Formulae: Genealogical and Interacting Particle Systems with Applications*. Probability and Its Applications. Springer, 2004. doi: 10.1007/978-1-4684-9393-1.

Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. Sequential Monte Carlo samplers. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(3):411–436, 2006.

Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. On adaptive resampling strategies for sequential Monte Carlo methods. *Bernoulli*, 18(1):252–278, 2012. doi: 10.3150/10-BEJ335.

Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. XGrammar: Flexible and efficient structured generation engine for large language models. *arXiv preprint arXiv:2411.15100v1*, 2024.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10764–10799, 2023.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. Grammar-constrained decoding for structured NLP tasks without finetuning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Horace He and Thinking Machines Lab. Defeating nondeterminism in LLM inference. Thinking Machines Lab: Connectionism, 2025. https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/.

Seong-Hwan Jun and Alexandre Bouchard-Côté. Memory (and time) efficient sequential Monte Carlo. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pp. 514–522, 2014.

Sooyeon Kim, Giung Nam, Byoungwoo Park, and Juho Lee. Improving constrained language generation via self-distilled twisted sequential Monte Carlo. *arXiv preprint arXiv:2507.02315*, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles*, 2023.

Alexander K. Lew, Tan Zhi-Xuan, Gabriel Grand, and Vikash K. Mansinghka. Sequential Monte Carlo steering of large language models using probabilistic programs. *arXiv preprint arXiv:2306.03081*, 2023.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can LLM already serve as a database interface? a big bench for large-scale database grounded text-to-SQLs. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2023.

Benjamin Lipkin, Benjamin LeBrun, Jacob Hoover Vigly, João Loula, David R. MacIver, Li Du, Jason Eisner, Ryan Cotterell, Vikash Mansinghka, Timothy J. O'Donnell, Alexander K. Lew, and Tim Vieira. Fast controlled generation from language models with adaptive weighted rejection sampling. In *Conference on Language Modeling (COLM)*, 2025. URL https://arxiv.org/abs/2504.05410. arXiv:2504.05410.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://arxiv.org/abs/2305.01210.

João Loula, Benjamin LeBrun, Li Du, Ben Lipkin, Clemente Pasti, Gabriel Grand, Tianyu Liu, Yahya Emara, Marjorie Freedman, Jason Eisner, Ryan Cotterell, Vikash Mansinghka, Alexander K. Lew, Tim Vieira, and Timothy J. O'Donnell. Syntactic and semantic control of large language models via sequential Monte Carlo. In *International Conference on Learning Representations*, 2025.

Daniel Lundén, Johannes Borgström, and David Broman. Correctness of sequential Monte Carlo inference for probabilistic programming languages. In *Programming Languages and Systems (ESOP)*, pp. 404–431. Springer International Publishing, 2021. doi: 10.1007/978-3-030-72019-3_15.

Shaan Nagy, Timothy Zhou, Nadia Polikarpova, and Loris D'Antoni. ChopChop: A programmable framework for semantically constraining the output of language models. *Proceedings of the ACM on Programming Languages*, 10(POPL):1905–1932, 2026. doi: 10.1145/3776708.

Brooks Paige, Frank Wood, Arnaud Doucet, and Yee Whye Teh. Asynchronous anytime sequential Monte Carlo. In *Advances in Neural Information Processing Systems*, pp. 3410–3418, 2014.

Kanghee Park, Jiayu Wang, Taylor Berg-Kirkpatrick, Nadia Polikarpova, and Loris D'Antoni. Grammar-aligned decoding. In *Advances in Neural Information Processing Systems*, 2024.

Gabriel Poesia, Oleksandr Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. Synchromesh: Reliable code generation from pre-trained language models. In *International Conference on Learning Representations*, 2022.

Isha Puri, Shivchander Sudalairaj, Guangxuan Xu, Kai Xu, and Akash Srivastava. Rollout roulette: A probabilistic inference approach to inference-time scaling of LLMs using particle-based Monte Carlo methods. *arXiv preprint arXiv:2502.01618v5*, 2025.

Qwen Team. Qwen2.5-Coder-1.5B-Instruct model card, 2024. URL https://huggingface.co/Qwen/Qwen2.5-Coder-1.5B-Instruct. Pinned revision recorded in the artifact.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021.

Freda Shi, Daniel Fried, Marjan Ghazvininejad, Luke Zettlemoyer, and Sida I. Wang. Natural language to code translation with execution. *arXiv preprint arXiv:2204.11454*, 2022.

Hanshi Sun, Momin Haider, Ruiqi Zhang, Huitao Yang, Jiahao Qiu, Ming Yin, Mengdi Wang, Peter L. Bartlett, and Andrea Zanette. Fast best-of-N decoding via speculative rejection. In *Advances in Neural Information Processing Systems*, 2024.

Chenglong Wang, Kedar Tatwawadi, Marc Brockschmidt, Po-Sen Huang, Yi Mao, Oleksandr Polozov, and Rishabh Singh. Robust text-to-SQL generation with execution-guided decoding. *arXiv preprint arXiv:1807.03100*, 2018.

Nick Whiteley, Anthony Lee, and Kari Heine. On the role of interaction in sequential Monte Carlo algorithms. *Bernoulli*, 22(1):494–529, 2016. doi: 10.3150/14-BEJ666.

Brandon T. Willard and Rémi Louf. Efficient guided generation for large language models. *arXiv preprint arXiv:2307.09702*, 2023.

Kevin Yang and Dan Klein. FUDGE: Controlled text generation with future discriminators. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2021.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for transformer-based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 521–538, 2022.

Stephen Zhao, Rob Brekelmans, Alireza Makhzani, and Roger Baker Grosse. Probabilistic inference in language models via twisted sequential Monte Carlo. In *International Conference on Machine Learning*, 2024.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, 2024.

Tianyang Zhou, Somesh Jha, Mihai Christodorescu, Kirill Levchenko, and Varun Chandrasekaran. Verifier-guided code translation via meta-step decoding. *arXiv preprint arXiv:2605.17626*, 2026.

A ADDITIONAL ATTRIBUTION AND EVALUATION DETAILS

We separate agreement under shared randomness, numerical behavior on the accelerator, model work, and elapsed time. Each protocol below measures a specific aspect of the execution contract.

Corrected attribution protocol. We froze the attribution protocol before collecting data. It uses the pinned Qwen2.5-Coder-1.5B-Instruct revision, N=8, $\tau = 1$, top-p=1, and three seeds. UB, UB+evict, compact EPB, and compact EPB+evict share the unit-step clock, verifier rules, and one H100. We interleave arms within each task on MBPP-sanitized (256), HumanEval (164), BIRD Mini-Dev (500), and GSM8K (200). Task-set shuffling and sampling use separate seeds. We derive method seeds using SHA-256, a deterministic 256-bit hash function. Pinning the model revision ensures that every arm uses the same weights. Our primary measurement is physical row-forwards. We obtain uncertainty intervals by resampling tasks after averaging seeds. Resampling both tasks and seeds checks sensitivity to these two sources of variation. It does not measure variation across deployments. Because the processes for three seeds overlapped with an external process, this protocol supports comparisons of row work and returned outputs. It does not support latency attribution.

UB+evict minus UB measures work spent forwarding completed rows. Compact EPB+evict minus UB+evict isolates certified-zero removal after finished rows are removed. Comparing compact EPB without eviction against UB mixes two sources of savings. The runner applies the terminal factor as soon as a row terminates and removes the row if the factor is zero. A completed program that fails its visible test is one example. We do not count this removal as an early certified prune. Table 5 reports the consequence per cohort. On MBPP, cEPB without termination eviction uses 22.2% fewer rows than dense UB, with 6.72 against 8.00 mean rows per forward, 0.06 certified prunes and 3.00 compaction events per run. On the 738 of 768 runs with no certified prune it still uses 707 rows against 915, a 22.8% reduction. HumanEval and BIRD behave the same way, whereas on GSM8K the runs with no certified prune save only 2.5%, which leaves certified prunes as the main source of its 11.6% saving with 1.51 prunes per run. The retained counters record certified prunes and compaction events. They do not record the number of terminal-zero rows removed. The zero-prune subsets identify savings without early certificates, rather than an exact partition by removal cause. They support terminal-zero removal as a major source of the code saving. Table 4 therefore reports Δ_1 and Δ_2, for comparisons in which both +evict arms already remove every terminated row. Terminal-zero removal therefore cannot explain the additional Δ_2 saving.

Table 5: Compact eager-prune barrier (cEPB) against unit barrier (UB), both without termination eviction, per run (task and seed). cEPB compacts certified-zero rows and rows that terminate with zero weight. Certified prunes, compaction events, and mean rows per forward (dense UB: 8) are recorded per run. Terminal-zero removals are not recorded, so the last columns compare runs with no certified prune. UB and cEPB columns count row-forwards. Compact. counts compaction events and Rows/fwd is the mean batch size per model call.

	All runs			cEPB, per run			Runs with no certified prune		
Cohort	UB	cEPB	Saved %	Prunes	Compact.	Rows/fwd	Runs	UB	cEPB
MBPP-256	923	718	22.2	0.06	3.00	6.72	738/768	915	707
HumanEval-164	1669	1303	21.9	0.19	3.60	6.46	429/492	1627	1279
BIRD-500	718	560	22.0	0.38	5.36	6.49	1095/1500	685	547
GSM8K-200	1479	1307	11.6	1.51	2.29	7.13	261/600	1319	1287

Law audit and numerical boundary. Table 2 separates the formal claim from accelerator behavior. We compare empirical output frequencies with the exact target over 20,000 trials per configuration. Paired random streams test finite-N coupling, while each mutation removes one contract requirement. The corrected implementation also agrees in 3,000 paired CPU runs. The richer target in Table 10 adds repeated barriers, unequal termination, soft factors, and collapse. Because default 16-bit brain floating-point (bfloat16) kernels remain batch-shape dependent, compaction can change logits and draws despite slot-indexed streams. Section 5.2 and Appendix B measure this gap and remove it in the audit mode.

Verifier and evaluator boundaries. The Python verifier treats an exception in a completed top-level statement as absorbing but defers repairable visible tests to termination. Unsupported constructs and timeouts abstain. The SQL rule certifies only failures that no suffix can repair (Appendix I). The GSM8K predicate checks completed calculator annotations and records each first-zero certificate and remaining token budget (Appendix J). These are operational predicates. They make no claim of general program or mathematical correctness.

Generation-visible success and hidden evaluation remain distinct. EvalPlus scoring froze hashes, retained strings, and task/seed/arm keys before hidden execution. Infrastructure failures remain unknown. A run without positive terminal mass fails the weighted-draw endpoint regardless of its fallback. All comparisons use this convention, and intervals covering zero remain unresolved.

Evaluation scope. The tasks with matched hidden-test scores in all arms comprise 223 MBPP and 164 HumanEval tasks. We do not substitute visible-test correctness for those endpoints. Quality contrasts use the named baseline in each contrast. In particular, Δ_2 uses UB+evict, while Δ_1 uses dense UB. Holm correction accounts for the stated test families. It does not cover uncertainty across models or deployments.

Runtime and transfer evidence. The earlier particle-scaling grid separates language-model (LM), verifier, and residual time, but predates the clock repair. The shared-batch study in §5.4 is descriptive and does not enforce a hard row budget. Neither study establishes vLLM or SGLang throughput.

We have no matched external-library comparison. Differences in the feedback function, cache, stopping rule, environment, and work counter prevent direct attribution of a performance difference to row removal. The retained evidence does not establish a contract violation in another library. Our reimplementation uses the same semantic-unit boundaries as `genlm/control-iclr-2025` (`af2493e8`) and `genlm-control` (`533a36ec`) under our task and verifier. It does not execute their code. Our inference is limited to the models, tasks, particle widths, hardware, and resampling rules of these studies.

B ACCELERATOR COUPLING, CONFIRMATION AND SHARED-BATCH SERVICE

We froze the three studies' predictions before collecting their measurements. We recorded execution changes before collecting the affected measurements. Both hosts were shared with other processes, and requests for higher process priority were denied. We log host load to describe this contention, but those logs do not correct its effect on timing.

Accelerator coupling (§5.2). Proposition 1(ii) assumes that a row's next-token distribution does not depend on the other rows in its batch. We test this directly with eight slots and four MBPP prompts. Each run decodes 60 tokens using slot-indexed streams. One retains all rows, while the other removes rows at three steps. With default 16-bit brain floating-point (bfloat16, or bf16) kernels, 390 of 960 surviving slot-steps, each a token step for one logical slot, change their logits (by up to 30.7 in logit units) and 8 of 16 surviving slots emit different text. We locate three batch-dependent reductions: the tiling used to group operations in general matrix multiplication (GEMM), the mean used by root mean square normalization (RMSNorm), and the partition of keys and values used by the attention kernel. A fixed-tile GEMM alone (He & Thinking Machines Lab, 2025) still leaves 288 slot-steps different, and adding one-program-per-row RMSNorm and row-wise attention removes every difference (0 of 960).

We evaluate the complete 256-task MBPP cohort with four arms, $N{=}8$, seed 0, and slot-indexed streams. The GEMM-only intermediate probe covers 59 tasks. Replacing linear layers with fixed-tile GEMM in Triton, a language for writing GPU kernels, still changed the positive-weight population in 4 of the 53 tasks where UB retains positive weight among 59 evaluated, without changing any returned draw. PyTorch RMSNorm changed a fixed row in up to 14 of 64 rows when batch size changed. Attention changed 74 of 100 context/subset pairs because its key-value reduction depends on batch size and context length. Our audit mode therefore uses one Triton RMSNorm program per row. It also computes attention and log-softmax, the logarithm of normalized token probabilities, one row at a time. Fixed-token repeats differed in 0 of 3,200 cells. These layer-level and fixed-token counts survive only as aggregate diagnostics in the execution records, without per-trial tensors.

One cohort exception (task 407) also varied between repeats of UB because the GPU scan over the 151,936-token vocabulary has timing-dependent addition order. A sequential double-precision floating-point (float64) CPU scan removed this source of variation. We count identity on tasks where UB retains positive terminal weight. The full rerun produced identical populations and draws on 219 of 219 tasks. Identity means equal SHA-256 fingerprints of the positive-weight population and returned draw in all four arms. Under default kernels, 207 of 222 tasks where UB retains positive weight are identical by that definition and 220 by the returned draw alone. The remaining 13 tasks therefore differ only in unselected particles and 2 in the returned draw. Figure 2(b) plots both counts.

This configuration is an audit reference and does not serve as a production path. It removes batch parallelism from attention and sampling. We did not implement a batch-parallel invariant attention kernel. The linear-layer replacements apply both to *prefill*, the initial model pass over the prompt, and to subsequent token decoding. We process each prompt alone with the same tensor shape across arms. The complete default and deterministic-reference cohorts each record zero timeouts and zero unsupported-code neutral events.

Complete-reference overhead at $N = 8$ and $N = 64$ remains unmeasured. The retained batch probe timed 24-row decode steps, each including a cache copy, at 16.8 ms under default eager kernels and 37.1 ms with the fixed-tile GEMM and per-row RMSNorm replacements under eager attention, a ratio of 2.2. That configuration excludes the row-wise attention and the sequential CPU scan of the complete reference. This is partial-reference overhead at one measured shape. It does not determine or bound complete-reference overhead. Contended cohort wall times cannot supply the comparison either.

Confirmatory GSM8K and GSM-Hard. We use GSM8K test problems 201 through 500 in the order set by the task-seed shuffle, disjoint from the exploratory 200, and the corresponding GSM-Hard problems (Gao et al., 2023). Predictions P1 and P2 require negative Δ_2 on both cohorts. We report task-bootstrap intervals and sign-flip tests, which assess a paired difference by reversing its sign under the null hypothesis. Holm correction covers exactly these two tests. P3 predicts more certified zeros on the harder variants of the same problems. Each cohort/seed/arm uses a pooled run with admission threshold 256 and $N = 8$. Admission follows the runner's dataset order after filtering the frozen task set. Per-request row counts use the single-request convention, but adding padding before shorter prompts and pooling them into batches changes the floating-point execution path. This is not a bitwise execution-path replication or a wall-time comparison. The arithmetic verifier has no timeout mechanism. All three predictions held (Table 4, §5.3). The row-work difference on fresh GSM8K is -35 $[-49, -21]$, with Holm-adjusted $p < 0.0001$. On GSM-Hard it is -113 $[-136, -90]$, with $p < 0.0001$. Runs without positive terminal mass under UB+evict and cEPB+evict number 5 and 2 of 900 cells on fresh GSM8K and 66 and 50 on GSM-Hard. These descriptive counts do not establish equal collapse probabilities.

Shared-batch runtime. The throughput study and step-time curve use an H100 MIG 2g.20gb partition (20 GB, two of seven compute slices) on a second host. All requests from one arm share a physical batch and one forward pass per token step. Each request runs the unchanged single-request schedule through a proxy engine that forwards its rows to the shared batch. The proxy keeps the unit-step clock, the exactly-once update, and the parked-weight rule of Algorithm 1. The proxy samples all rows of a request from one generator in physical row order, and the pooled kernels see the batch shape. The slot-indexed stream and fixed-kernel premises are therefore not met, and Proposition 1(ii) does not cover this tier. We pad shorter prompts on the left, mask padding positions in attention, and supply explicit token positions. For resampling or compaction, we apply the same source-row mapping to all of a request's tensors. A preflight decoded three prompts while duplicating and removing rows and admitting one request late. In 32-bit floating point (float32), every greedy token, the token with highest probability, matched single-request decoding and logits differed by less than 5×10^{-4}. In bf16, one near-tie changed the greedy token. The throughput protocol covers full GSM8K-fresh and MBPP cohorts, each arm, N=8, admission thresholds $C \in \{64, 128\}$, and three seeds. Its primary endpoint is tasks per minute. Tasks per 1,000 forward steps is the other primary endpoint. It removes wall-time variation and leaves occupancy differences in place. We predicted UB+evict above UB in every cell, cEPB+evict above UB+evict on average for GSM8K, and no direction for MBPP.

Admission checks current occupancy plus N, without reserving space for later resampling expansions. Dense UB stays at or below C. Other arms reach 76 and 142 rows. The cells therefore share an admission rule without an enforced row budget. Table 6 reports absolute throughput, seed ratios, mean resident rows, and peaks. Three-seed ranges are not confidence intervals. The table also reports tasks per 1,000 pooled row-steps. This divides completed tasks by resident rows summed over steps. It normalizes measured model-row work, but batch composition can still alter numerical trajectories and logical decisions. It does not replace a capacity-matched comparison. Under that endpoint UB+evict completes 1.55× (GSM8K) and 1.88× (MBPP) UB's tasks at $C = 64$, and cEPB+evict completes 1.04× and 0.97× UB+evict's, with the same direction at $C = 128$. On MBPP at $C = 64$, cEPB without termination eviction achieves 1.16× UB's tasks per minute. It records 0.06 certified prunes per request and 744 rows per request against UB's 941. These counts are consistent with savings from compacting terminal-zero rows. The retained counters do not provide a complete event-level attribution.

Table 6: Shared-batch tasks/minute by seed and three-seed mean, tasks per 1,000 pooled row-steps (three-seed mean, normalized by measured model-row work), mean resident rows, and peak occupancy. Thresholds are admission settings and do not act as hard row limits. cEPB compacts certified-zero rows and rows that terminate with zero weight. Pruning/eviction compares cEPB+evict with UB+evict. One row-step evaluates one resident row once. n/a means no occupancy ratio is defined for these throughput comparisons. UB is unit barrier and cEPB is compact eager-prune barrier.

Arm	Seed 0	Seed 1	Seed 2	Mean	Tasks/1k row-steps	Mean rows	Peak rows
GSM8K, admission threshold 64							
UB	36.1	35.2	35.0	35.4	0.71	63.1	64
UB+evict	43.1	43.3	42.8	43.1	1.09	57.7	69
cEPB	38.3	37.5	37.8	37.9	0.80	59.1	73
cEPB+evict	44.8	43.4	44.2	44.1	1.13	57.3	68
Pruning/eviction ratio	1.04	1.00	1.03	1.03	1.04	n/a	n/a
GSM8K, admission threshold 128							
UB	36.5	35.1	35.0	35.5	0.71	123.2	128
UB+evict	43.2	42.5	42.4	42.7	1.10	113.2	133
cEPB	37.7	37.2	37.6	37.5	0.80	118.0	133
cEPB+evict	44.2	42.6	43.6	43.4	1.13	114.4	133
Pruning/eviction ratio	1.02	1.00	1.03	1.02	1.03	n/a	n/a
MBPP, admission threshold 64							
UB	67.8	64.6	61.6	64.7	1.06	62.8	64
UB+evict	73.3	81.0	77.2	77.2	2.00	57.3	76
cEPB	73.1	76.9	75.2	75.1	1.34	58.1	75
cEPB+evict	84.6	66.7	70.9	74.1	1.95	57.3	73
Pruning/eviction ratio	1.15	0.82	0.92	0.97	0.97	n/a	n/a
MBPP, admission threshold 128							
UB	65.2	65.2	67.2	65.8	1.07	121.0	128
UB+evict	85.8	88.1	81.7	85.2	2.01	105.0	135
cEPB	70.7	78.0	75.5	74.8	1.33	112.6	142
cEPB+evict	73.5	71.0	77.9	74.1	1.95	109.2	135
Pruning/eviction ratio	0.86	0.81	0.95	0.87	0.97	n/a	n/a

Throughput ratios and work. Adding pruning to eviction gives GSM8K throughput ratios of 1.03× and 1.02× at $C = 64$ and 128. For MBPP, the corresponding ratios are 0.97× and 0.87×.

At $C = 64$, we also compare UB+evict with dense UB. Per 1,000 forward steps, the GSM8K and MBPP ratios are 1.42× and 1.72×, respectively (Figure 2(d)).

For the additional effect of pruning, we compare cEPB+evict with UB+evict in tasks per 1,000 pooled row-steps, where one row-step evaluates one resident row once. At thresholds $C = 64$ and 128, the

GSM8K ratios are 1.04× and 1.03×, and the MBPP ratios become 0.97× and 0.97×, respectively at $C = 64$ and 128.

Recorded time components. Gathering assembles request rows into a shared batch. The global forward timer includes this step, model execution, and device synchronization. The separate gathering timer is nested within it, so the two times cannot be added. Prefill is timed separately. We call the total elapsed time to finish the cohort its *makespan*. Table 7 reports time outside the recorded components as uninstrumented runtime overhead. It includes coordination and waiting and is not a measured CPU/GPU split. Per-request timers overlap and cannot be summed into this partition. For dense UB, the forward timer exceeds the single-request decode step on the same partition by 3.2 to 3.8 times on GSM8K (54.8 against 17.1 ms at $C = 64$) and 2.1 to 2.4 times on MBPP (Table 7). The gap suggests runtime overhead. Our timers do not isolate Python, synchronization, or accelerator idle time. A matched threshold also does not hold request concurrency fixed.

Table 7: Recorded time components, averaged over three seeds. Forward includes gathering request rows into a batch and synchronizing the device. Bare step is the single-request decode step on the same partition at the threshold's row count, and Ratio divides the forward timer by it. Shares are percentages of makespan, the elapsed time to finish the cohort. Prefill is the initial model pass over a prompt. Residual includes uninstrumented execution and waiting.

Arm	Forward ms/step	Bare step ms	Ratio	Forward (%)	Prefill (%)	Residual (%)
GSM8K, admission threshold 64						
UB	54.8	17.1	3.2	72.6	1.9	25.4
UB+evict	56.5	17.1	3.3	64.4	2.4	33.2
cEPB	53.3	17.1	3.1	71.5	2.0	26.5
cEPB+evict	56.4	17.1	3.3	63.9	2.4	33.7
GSM8K, admission threshold 128						
UB	103.3	27.0	3.8	70.3	2.3	27.4
UB+evict	109.1	27.0	4.0	62.6	2.9	34.6
cEPB	103.6	27.0	3.8	68.5	2.4	29.1
cEPB+evict	111.3	27.0	4.1	62.2	3.0	34.8
MBPP, admission threshold 64						
UB	35.4	17.1	2.1	57.1	2.5	40.4
UB+evict	39.1	17.1	2.3	43.9	3.1	53.0
cEPB	35.1	17.1	2.0	56.2	3.0	40.8
cEPB+evict	38.9	17.1	2.3	43.0	3.1	53.9
MBPP, admission threshold 128						
UB	65.5	27.0	2.4	55.5	3.0	41.5
UB+evict	67.6	27.0	2.5	45.5	4.0	50.5
cEPB	65.5	27.0	2.4	54.2	3.5	42.2
cEPB+evict	68.7	27.0	2.5	39.7	3.4	56.8

Certificates, power and zero mass. All 1,822 rejected annotations are unequal under strict rational arithmetic. With the operational tolerance, 1,820 remain rejected and two are accepted by the rational reference. Appendix C gives both events and their residuals. These outcomes support a claim about the declared floating-point predicate, not exact arithmetic-semantic soundness. Statistical *power* is the probability of detecting an effect of a specified size under the planning assumptions. At 80% power and an unadjusted two-sided 5% significance level, the earlier analysis estimates detectable Δ_2 quality differences of 3.7 (MBPP), 3.8 (HumanEval), 3.1 (BIRD) and 6.1 (GSM8K) points. For code these use visible-test correctness. The held-out EvalPlus scores do not enter them. Table 8 instead matches Table 4's quality endpoints. For task i, let d_i be the cEPB+evict minus UB+evict quality difference averaged over the three seeds, and let s_d be its sample standard deviation across tasks. We approximate the required task count using the normal distribution. Writing $\lceil \cdot \rceil$ for rounding upward to the next integer, the planning count is $\lceil [(1.959964 + 0.841621)s_d/0.02]^2 \rceil$ tasks for a two-percentage-point effect at two-sided 5% level and 80% power. The constants are the corresponding normal quantiles. This holds the observed task variance and three-seed design fixed. It is not guaranteed power on new tasks.

Table 8: Quality-endpoint planning estimates at 80% power and unadjusted two-sided 5% level, holding observed task variance fixed. Minimum detectable effect (MDE) is in percentage points. Power is the probability of detecting the stated effect under the planning assumptions. Required counts are distinct tasks with three seeds each.

Endpoint	Observed tasks	MDE	Tasks for 2 points
MBPP (held-out)	223	4.6	1,196
HumanEval (held-out)	164	5.3	1,146
BIRD (execution)	500	3.1	1,236
GSM8K (answer)	200	6.1	1,884

We use McNemar's exact test to compare paired binary collapse outcomes. Runs with no positive terminal mass under exactly one of UB+evict and cEPB+evict split 2 to 6 on GSM8K (exact McNemar $p = 0.29$) and 30 to 31 on MBPP ($p = 1.00$).

C Complete status and verifier audits

All-task status. Positive mass means at least one particle has positive selection weight. It does not imply correctness on hidden tests. Collapse means every weight is zero. All positive means every arm retains positive mass, all collapsed means every arm collapses, and mixed means the arms disagree. The four arms are unit barrier (UB), compact eager-prune barrier (cEPB), and each with finished-row eviction (+evict). Identity requires equal SHA-256 fingerprints of both the positive-weight population and the returned draw. The earlier conditional counts use tasks where dense UB retains positive weight. General matrix multiplication (GEMM) is the only operation controlled in the 59-task intermediate probe. Fixed reductions also control normalization and attention. The full reference additionally fixes the cumulative-probability scan used for sampling.

Mode	Tasks	All positive	All collapsed	Mixed
Default	256	221	34	1
GEMM only	59	53	6	0
Fixed reductions	256	219	37	0
Full reference	256	219	37	0

There are no missing status records in these cohorts. The next table joins the complete default and reference cohorts by task. Rows give the default status and columns give the reference status.

Default status / reference status	All positive	All collapsed	Mixed
All positive	212	9	0
All collapsed	7	27	0
Mixed	0	1	0

On the 212 tasks with positive mass in every arm under both modes, all-arm identity holds on 200 default tasks and 212 reference tasks. This shared subset is the common support for the comparisons below. Each row compares one arm across modes.

Arm	Population and draw	Draw alone
UB	124	176
UB+evict	124	176
cEPB	124	176
cEPB+evict	124	176

These comparisons use the same 212 tasks. They measure agreement under shared randomness, rather than distributional distance. The reference may define a different numerical proposal.

Arithmetic boundary cases. All 1,822 rejected annotations are unequal under strict rational arithmetic. With absolute and relative tolerance 10^{-6}, the rational reference rejects 1,820 and accepts two events (0.11%). Both contain `<<3,333.33*12=40,000>>`. The residual is the

absolute difference between the two sides, and the threshold is the largest difference the check accepts. Under rational evaluation, both equal $1/25 = 0.04$. Floating-point evaluation gives residual 0.040000000000873115 and threshold 0.04, so the operational check rejects. The two retained events are: Seed 2, task index 168, cEPB, event 2. Seed 2, task index 168, cEPB+evict, event 2. Thus an absorbing zero is sound relative to the declared operational target, whose terminal constraint excludes operationally rejected annotations. It is not a guarantee of exact arithmetic-semantic validity. The confirmatory records contain prune counts but no certificate strings, so we cannot repeat this replay for those cohorts.

Where pruning becomes possible. The first-zero unit index identifies the completed unit at detection. Counts by index 1 through 9 are 646, 341, 350, 256, 139, 54, 31, 3, 2. The mean remaining token budget is 219.4, with median 234 and range 11 to 308. Remaining budget describes opportunity. It does not measure avoided forwards or partition savings by removal cause.

HumanEval timeout sensitivity. We keep all 164 tasks as the primary cohort. A secondary analysis excludes the six tasks with any recorded timeout in any arm or seed, leaving the same 158 tasks for all arms. We average the three seeds within each task and use 10,000 task-bootstrap draws, matching the primary procedure. Intervals below are marginal 95% intervals. The contrast Δ_1 subtracts UB from UB+evict, Δ_2 subtracts UB+evict from cEPB+evict, and total subtracts UB from cEPB+evict. Row-forwards count model-row evaluations. Hidden quality is the held-out EvalPlus accuracy of the returned draw, with differences in percentage points.

Contrast	Row-forwards	Hidden quality (points)
Δ_1	-713.6 [-767.3, -660.8]	0.84 [-1.90, 3.59]
Δ_2	-10.7 [-34.9, 13.8]	-1.69 [-5.49, 2.11]
Total	-724.3 [-779.0, -669.7]	-0.84 [-4.01, 2.32]

Exclusion conditions on recorded outcomes and does not create a new unbiased cohort. This sensitivity analysis does not establish equivalence.

D STOPPING-TIME DEFINITIONS AND OCCUPANCY IDENTITIES

We formalize the two update clocks and then relate resampling to verifier reuse. The clock determines when positive evidence enters the population. Ancestor occupancy, the number of distinct ancestors selected at resampling, determines how many prefixes may need verification. Unlike physical row occupancy, it counts selected ancestors rather than resident model rows.

Barrier stopping time. Let t denote the batched token step, $i \in \{1, \dots, N\}$ a logical slot, $b \geq 0$ an integer barrier level, $M_t^{(i)}$ the completed-unit count of slot i at that step, and τ_b the step at which barrier level b fires, with $\tau_0 = 0$. The infimum below selects the first step satisfying the barrier condition. UB checks at

$$\tau_{b+1} = \inf\{t > \tau_b : \min_{i \text{ active}} M_t^{(i)} \geq b + 1\}, \tag{2}$$

with terminated particles treated as ready. Let $\mathcal{F}_t$ denote the history observed through step t. This history determines whether the barrier is ready, so its firing time is a stopping time for the particle-system filtration $(\mathcal{F}_t)_{t \geq 0}$. Fast particles may hold more than $b + 1$ completed units when the slowest reaches the barrier. Their weight update then telescopes across several unit factors.

Bounded-lag stopping time. Lag-bounded EPB (LB-EPB) permits positive updates before the common barrier and lies outside the finite-N coupling. Here a timeout bounds how long an observed score can wait for application, not how long verifier execution can run. Its clock belongs to an *unassimilated observation*, a score not yet applied to the population. We do not attach this clock permanently to a barrier. Let j index population checkpoints and κ_j be the time of the jth checkpoint. The time of the first new positive observation is $a_j = \inf\{t > \kappa_j : \text{a live row produces a new positive observation}\}$, and β_j is the next common-unit barrier after κ_j. With lag bound B and maximum token horizon T, LB-EPB assimilates all observed scores at

$$\kappa_{j+1} = \min\{\beta_j,\ a_j + B,\ T\}. \tag{3}$$

All per-row clocks clear after any checkpoint. Repeated timeouts can therefore occur while the barrier index is unchanged, and every assimilated positive observation is at most B batched token steps old. $B = \infty$ recovers EPB.

We next count how many different ancestors a resampling step selects. Here we use normalized weights, unlike the unnormalized weights in Algorithm 1. Let $w = (w_1, \dots, w_N)$ be those weights before resampling, D the number of distinct selected ancestors, and $\mathbb{E}[D \mid w]$ its expected value conditional on those weights.

Proposition 2 (Exact multinomial occupancy)**.** *Given normalized pre-resampling weights $w_{1:N}$ and N multinomial offspring, the number D of occupied ancestors satisfies*

$$\mathbb{E}[D \mid w] = \sum_{j=1}^{N} \left[1 - (1 - w_j)^N\right] \leq N. \tag{4}$$

Each selected ancestor may require a verifier result. A pure prefix store performs at most one new execution per distinct prefix absent from the cache. We use $O(1)$ for a term bounded independently of N. No universal bound of the form $\mathbb{E}[D] \leq \mathrm{ESS} + O(1)$ holds: for $w_1 = 1/2$ and $w_j = 1/[2(N-1)]$ otherwise, ESS approaches four while Equation 4 grows linearly in N, denoted $\Theta(N)$.

E Stopping-time details and exact benchmark

We first explain how changing the base distribution affects the weights. We then prove consistency and finite-N equivalence, connect these arguments to parking and compaction, and describe the exact benchmark.

Let r be a temperature-independent base distribution that assigns positive probability to exactly the same sequences as q_τ. Write π_r for the resulting target and $W(x)$ for the importance weight that corrects a sequence drawn from q_τ. They are

$$\pi_r(x) \propto r(x)H(x)\prod_m g_m(u_{1:m}), \qquad W(x) = \frac{r(x)}{q_\tau(x)}\,H(x)\prod_m g_m(u_{1:m}), \tag{5}$$

with $\log\big(r(x)/q_\tau(x)\big)$ accumulated token by token. Because every reported run sets $r = q_\tau$, the proposal ratio is one.

Our filtration $\mathcal{F}_t$, the information available through token step t, contains every particle prefix, active flag, weight, completed-unit count, verifier response and resampling draw observed through batched token step t. A Periodic schedule checks at fixed token intervals, and its grid is therefore deterministic. The Event schedule checks after a completed-unit key changes or a particle terminates. Its trigger is $\mathcal{F}_t$-measurable, meaning that we can decide whether it has occurred using only the information available by step t. The barrier event in Equation 2 is likewise measurable, which makes its first hitting time τ_b a stopping time. EPB uses the same population-update grid. Its additional observations are adapted, meaning available at their observation times, and acting on an absorbing zero removes only a trajectory whose future target mass is identically zero. At τ_b, particles may have different $M_t^{(i)}$. Because each uses its own cumulative potential and previous score, the argument assumes no equality of unit counts.

In token time the sampler is a Feynman-Kac model, a weighted distribution over successive particle paths. For particle i, let $u_{1:m}^{(i)}$ denote its first m units, $M_t^{(i)}$ its unit count at step t, and $w_t^{(i)}$ its weight then. The token-generation transition is $q_\tau(\cdot \mid x_{<t})$. Its cumulative potential is $G_t^{(i)} = \prod_{m \leq M_t^{(i)}} g_m(u_{1:m}^{(i)})$, the terminal factor H entering once at the horizon. The recursion $w_t^{(i)} \propto w_{t-1}^{(i)} G_t^{(i)}/G_{t-1}^{(i)}$ is evaluated only at checkpoints. The potential ratio is one when a row has completed no new unit. Thus, choosing checkpoints from the population history does not add a factor to its weight. Here C is a uniform score bound, distinct from the admission threshold C used in the runtime study, and K_T bounds the number of units through token horizon T. More precisely, assume $0 \leq g_m(u_{1:m}) \leq C$ and $M(x) \leq K_T < \infty$ for all reachable prefixes up to the fixed horizon T, with C and K_T independent of N. Since our segmenters emit nonempty units, the token horizon also bounds their number. Let J be the number of population checkpoints after initialization, indexed by

$j \in \{1, \dots, J\}$. We write their successive times as $0 = \kappa_0 < \kappa_1 < \cdots < \kappa_J \le T$. For one lineage, the sequence of states inherited through successive ancestors, we omit the particle superscript. If its previous cumulative potential is positive,

$$\frac{G_{\kappa_j}}{G_{\kappa_{j-1}}} = \prod_{m=M_{\kappa_{j-1}}+1}^{M_{\kappa_j}} g_m(u_{1:m}) \le \max(1, C)^{K_T}.$$

The empty product is one. The bound follows by cancellation of previously applied factors. It does not divide arbitrary bounded cumulative quantities. Including terminal $H \in \{0, 1\}$ cannot increase it. A zero increment deactivates the row. It never subsequently divides by its zero cumulative potential. A positive target normalizer defines the limiting normalized measure, but does not preclude a finite-N collapse. The incremental potential applied to row i at checkpoint j is $G^{(i)}_{\kappa_j}/G^{(i)}_{\kappa_{j-1}}$. Terminal-only H is applied once if it has not already been included. Their product telescopes to $G^{(i)}_T$. Let φ be a bounded test function on particle states and $\mathcal{R}_N(\varphi)$ its unweighted empirical average after resampling. In the following display, w_i denotes the normalized weight of particle i, rather than the unnormalized weight used in Algorithm 1. Its state is x_i, and $\xrightarrow{\mathbb{P}}$ denotes convergence in probability, with $\mathbb{P}$ the probability law of the sampler:

$$\mathcal{R}_N(\varphi) - \sum_{i=1}^{N} w_i \varphi(x_i) \xrightarrow[N\to\infty]{\mathbb{P}} 0 \quad \text{conditionally on } \mathcal{F}_{\kappa_j}.$$

This excludes, for example, a conditionally unbiased rule that copies a single random ancestor into all N rows: its conditional error need not vanish. Multinomial resampling satisfies the display by its conditional variance bound. Our consistency claim is therefore instantiated by the implemented multinomial sampler. The systematic and stratified occupancy diagnostics do not extend the theorem without separately checking the displayed condition for their particle ordering. Because the horizon is finite, the number of checkpoints is finite. Bounded adapted potentials and the conditional law-of-large-numbers induction for adaptive SMC therefore give consistency of normalized weighted averages whenever the target normalizer is positive (Del Moral et al., 2006, Lundén et al., 2021). A pure store replaces a deterministic evaluation by the same stored value and leaves this argument unchanged. If the normalizer estimate is zero because every row has zero weight, the algorithm reports collapse. Resetting the weights would instead change the target and is explicitly excluded.

For the finite-N statement in Proposition 1, we assign each logical slot its own proposal stream, with one draw from the uniform distribution per slot and token step. The induction state comprises each slot's text, proposal log-probabilities, live/parked/zero status, weight, cumulative potential, completed-unit count, and finalized flag, together with the common barrier level. Resampling copies this state from the selected ancestor, resets the offspring's weight, and assigns its stream by offspring position. Siblings therefore use distinct streams. Copying a finalized parked ancestor preserves both its eligibility and its no-rescoring state. We compute ESS and the resampling cumulative distribution function (CDF) in slot order. We couple EPB and UB on those streams through each barrier. When EPB observes an absorbing zero, every continuation of that row has terminal mass zero. Relative to the next threshold $b + 1$, a newly zeroed row is already ready: the zero was observed only after completing that unit, or after termination, which is ready by definition. Removal therefore leaves the barrier predicate unchanged. The corrected rule increments b once at this checkpoint and never at resampling. Equal incoming clocks then yield equal outgoing clocks, even if surviving particles have completed several units. Induction starts at $b = 0$ and preserves both readiness and the next threshold. Jumping to the minimum surviving unit count would break this argument. A tombstone cannot change another particle's proposal law, and ordinary UB assigns the removed row zero weight at the next barrier.

The normalized nonzero weight vectors and conditional ancestor distributions therefore agree at the barrier. A shared random draw selects the same nonzero ancestors, after which the logical proposal streams agree again. Induction over barriers establishes the coupling. At T, a common weighted random draw applied to the equal normalized measures returns the same terminal draw. Tensor compaction changes only physical indices and thus preserves the abstract coupling. Accelerator kernels, however, may consume a different global random-number-generator (RNG) sequence after a batch-shape change. With default kernels the GPU attribution is therefore an implementation comparison. With the batch-invariant, deterministic kernels of §5.2 the premises hold and all arms return identical draws.

Clock defect, repair and evidence versions. The earlier implementation computed the next level from the surviving rows. This was an implementation defect and went beyond an omission in the pseudocode. With $N = 3$, a zero at unit 1 and two survivors at unit 2, UB set $b = 1$ while eager removal set $b = 2$. At the next token step, a factor of 9 on one survivor gave UB weights $(0, 9, 1)$ and $\mathrm{ESS} = 100/82 < 3/2$. UB therefore resampled at step 2 while the eager schedules waited until step 3. Our deterministic regression reproduces both paths. With the corrected unit-step clock, UB, UB+evict, EPB, compact EPB and compact EPB+evict all use $b = 1$ then $b = 2$, normalize the survivors to $(0.9, 0.1)$, and make the same ancestor draws at step 2. Tests cover eight seeds and preserve the legacy mode to make the defect reproducible. The compact variants also no longer advance the clock after selecting ancestors.

A new CPU re-audit checks 500 paired seeds for each $N \in \{4, 16, 64\}$ and each unit-duration scale in $\{1, 4\}$: all 3,000 corrected UB and EPB pairs agree in output, normalizer estimate and resampling count. At scale 1, 3,000 comparisons with the separately written sampler agree in output and normalizer estimate (absolute/relative tolerance 10^{-12} for the latter). These finite checks support the corrected implementation. They do not certify all targets. Because the corrected GPU attribution uses the same unit-step clock in all four arms, its Δ_2 no longer includes a clock change. Historical GPU rows remain versioned and separate.

Eviction of terminated rows. Park on termination removes a row from the batch and from every KV layer, whatever its verdict, and keeps its logical state (text, log-probabilities, weight, cumulative potential, finalized flag) in a parked slot. A terminated row receives no token that any potential reads: the dense runner computes its later tokens and discards them. Its weight at every later checkpoint is therefore a function of the parked state alone. It follows that the barrier updates, the normalized weight vector and the conditional ancestor distribution at each barrier are those of the dense runner. A parked ancestor selected at resampling yields a parked copy, since a finalized row is never extended, and the proposal law of a live row is unchanged because attention is per row. Given proposal streams indexed by logical slot, the induction above therefore couples UB with UB+evict and compact EPB with compact EPB+evict. A run without positive terminal mass fails the endpoint under every arm, and no fallback string enters the comparison (Appendix A). CPU regression tests instantiate such streams without floating-point model kernels. A two-particle example shows why retaining the weight is essential. Both rows reach the barrier with unnormalized weights $w_1, w_2 > 0$, and row 1 has terminated. Parking resamples from $(w_1, w_2)/(w_1 + w_2)$, and row 1 is then returned at the horizon with probability $w_1/(w_1 + w_2)$. Freeing the row and merely recording its string resamples from $(0, w_2)/w_2$, returning it with probability zero. A terminated row with positive weight must therefore stay in the weight vector and the ancestor distribution until the terminal draw. Only its tensors and cache entries may be freed (0.120 total variation, 6.2% more work, Table 2).

Persistent finalization example. A positive particle terminates at the first checkpoint while another completes three later units. The first terminal evaluation sets its finalization flag to $f_i = 1$. Subsequent checkpoints leave its potential and weight unchanged without another verifier call. If resampling copies that ancestor into two parked offspring, the copies both inherit $f_i = 1$ and remain eligible for weighted selection without rescoring. The regression test for terminal scoring across later checkpoints checks both cases. Since H is binary, repeated evaluation of the same positive terminal state would not itself prove weight bias. The flag specifies exactly-once execution and agrees with the runtime's persistent `fin` guard.

For LB-EPB, a_j in Equation 3 is the first newly observed positive score after checkpoint κ_j. The event $\{a_j \le t\}$ is $\mathcal{F}_t$-measurable, and $a_j + B$ is therefore a stopping time for deterministic B. Its minimum with the next common barrier and terminal horizon is also a stopping time. Assimilating the cumulative-potential ratio clears every pending clock. If the semantic barrier has not advanced, a later observation starts a fresh clock and can cause a second timeout. This construction is a recurring clock and does not behave as a one-shot timer.

The exact study in Table 10 draws up to four units autoregressively from $\{\mathtt{A}, \mathtt{BB}\}$, whose token lengths are one and two. It terminates early on $(\mathtt{BB}, \mathtt{A}, \mathtt{BB})$, assigns an absorbing zero to prefixes beginning $(\mathtt{A}, \mathtt{A})$, assigns a terminal-only zero to completions ending $(\mathtt{BB}, \mathtt{BB})$, and applies strictly positive depth- and transition-dependent factors elsewhere. We enumerate its nine positive-mass terminal strings before simulation. For each schedule, we use identical proposal and resampling randomness when comparing runs with and without the prefix cache. The experiment aborts if any paired output differs. We retain weights in log space and normalize after subtracting the largest finite

log weight. An integrity test verifies finite normalized probabilities for log weights below -10^4. The exact study counts logical proposals of active particles and has no physical batch. A particle that terminates with zero weight becomes inactive under every schedule, and no compaction rule enters the Δ work column of Table 2 or the token counts of Table 10. Table 9 reports the posterior-fidelity diagnostic on the same family of targets, and the weight family stated with Proposition 2 rules out an ESS-only execution bound.

Table 9: Posterior-fidelity diagnostic at $N = 32$ (10,000 trials). Terminal importance sampling (IS) weights complete samples at termination using the full potential product. Uniform-valid-of-N uses hard validity only. TV means total variation distance and ESS means effective sample size. The moment is the expected fraction of units equal to `A`. Draw error is the absolute error in this moment across returned draws. Moment RMSE is the root mean squared error of the weighted population estimate. The target normalizer is Z. Its relative RMSE measures normalizer error. n/r means not recorded. Event updates at completed units and LB-EPB is lag-bounded eager-prune barrier with lag B.

Method	TV	Draw error	Moment RMSE	Z rel. RMSE	ESS
Uniform-valid-of-N	0.556	0.128	n/r	n/r	n/r
Terminal IS	0.073	0.017	n/r	n/r	n/r
Event	0.030	0.006	0.037	0.298	21.5
Unit barrier	0.021	0.005	0.038	0.297	22.4
Eager-prune barrier	0.025	0.004	0.037	0.301	22.4
LB-EPB ($B = 0$)	0.022	0.004	0.036	0.301	21.3

Table 10: Richer exact target at $N = 32$ (10,000 paired trials): repeated barriers, soft factors, absorbing and terminal zeros, unequal termination. Unit-barrier (UB) and eager-prune barrier (EPB) have zero mismatches in 50,000 pairs across $N = 4, 8, 16, 32, 64$. TV is total variation distance and Z the target normalizer. Resamplings, logical token proposals, and verifier executions are means per run. Event updates at completed units. LB-EPB allows positive updates within lag B of observation.

Schedule	TV	Relative Z bias	Resamplings	Logical tokens	Executions
Event	0.0310	-0.0037	1.45	201.4	19.7
Unit barrier	0.0268	-0.0046	1.44	200.1	17.5
Eager-prune barrier	0.0268	-0.0046	1.44	200.1	18.0
LB-EPB ($B = 0$)	0.0316	-0.0037	1.44	201.4	19.7

Implementation obligations of Proposition 1. Units are append-stable, zeros are append-irreparable, and the logical barrier clock is shared in a coupling comparison. Terminal H is applied once. Dead rows remain at $\log w = -\infty$ without evaluating $-\infty - (-\infty)$. A live row may become finalized-positive at termination or a zero tombstone. A finalized row is never updated again. Only an eligible ESS-triggered resampling creates new live descendants. Algorithm 1 makes these states explicit, and unit tests exercise their boundary cases. On the corrected exact-target fixture, Table 2 deletes one clause of the verifier or runtime contract per row, apart from the declared LB-EPB variant.

F Implementation and Security

The LM engine stores attention keys and values in a Hugging Face `DynamicCache`. *Prompt prefill*, the initial model pass over the input prompt, occurs once per question. Resampling calls `reorder_cache` to copy cache rows for the selected ancestors. Compacting schedules select surviving rows before the next model call. The dense EPB ablation retains zero-mass rows. Because both logical proposals and physical row-forwards are counted, the comparison measures work removed by selection directly and does not infer it from particle flags.

The SQL verifier uses a read-only SQLite uniform resource identifier (URI), a callback that stops queries exceeding the time budget, and a `sqlglot` tokenizer that splits SQL text into syntactic elements. Benchmark scoring executes each prediction separately and shares the deterministic gold-query result once per question across methods. Evaluation queries contribute neither verifier nor

timing counters. The Python verifier keeps namespaces in worker processes, copies interpreter state using *copy-on-write*, where a child process shares memory until either process changes it. It executes each completed statement once and applies the visible assertion only at termination.

A *call* requests a prefix potential. A prefix with no completed unit returns the neutral factor 1 without execution. A store hit also costs no execution. On a miss, the stateless worker executes all completed units. The stateful worker executes only the unseen suffix and stops at the first zero. Executions can therefore be fewer or greater than calls, which is why the store tables report both counters.

An abstract syntax tree (AST) gate over generated code rejects unlisted imports, direct dynamic evaluation, and double-underscore ("dunder") attribute access. The Seatbelt profile applies on macOS only. Elsewhere, the runner refuses unsandboxed Python unless the caller acknowledges that an external container provides isolation.

G Transfer and comparison details

The shared-batch runtime (§5.4) serves requests in one physical batch but is not a production server with paged KV memory. Memoization reduces actual verifier executions under a fixed schedule (Table 11). To measure its interaction with eviction, we would need matched runs with each mechanism independently enabled and disabled. A matched external-library comparison would also need common stopping rules, inputs to the feedback function, caching, and environment.

Future-discriminator controlled generation (FUDGE) (Yang & Klein, 2021), token-level SMC (Lew et al., 2023) and semantic SMC (Loula et al., 2025) establish the controlled-generation setting. Our Event implementation updates at completed semantic units. Learned twists, functions that steer sampling toward promising futures, modify the proposal (Zhao et al., 2024, Kim et al., 2025), whereas our posterior table covers fixed-potential schedules. Adaptive weighted rejection sampling (Lipkin et al., 2025) queries candidate next tokens, whereas our verifier executes completed spans. Its query count therefore measures a different operation from our execution counter.

Classical memory-efficient SMC stores ancestry compactly (Jun & Bouchard-Côté, 2014). We remove terminal or zero rows while preserving the state used by weighted selection. Orca's finished-request removal (Yu et al., 2022) therefore covers only the physical half of our obligation. A common barrier enables our finite-N coupling. Asynchronous SMC (Paige et al., 2014) is preferable when synchronization dominates but changes the resampling mechanism. Production runtimes based on vLLM's PagedAttention or SGLang (Kwon et al., 2023, Zheng et al., 2024) would also require logical slots, parked weights and resampling state to remain independent of KV locations. Execution-guided and constrained decoding (Wang et al., 2018, Shi et al., 2022, Willard & Louf, 2023, Geng et al., 2023, Dong et al., 2024, Park et al., 2024, Poesia et al., 2022, Scholak et al., 2021), reward models (Puri et al., 2025) and verifier search (Brandfonbrener et al., 2024, Agrawal et al., 2023) supply other feedback mechanisms. Uniform-valid-of-N provides a cost reference that samples a different distribution. ChopChop (Nagy et al., 2026) and meta-step decoding (Zhou et al., 2026) suggest stronger structural certificates that we have not tested.

H Additional empirical scope limits

The corrected full-cohort rerun used three concurrent seed workers, and another process was observed during part of the run. It therefore provides no isolated latency estimate. Historical timing grids remain explicitly versioned. Quality contrasts pair the named arms by task and seed. The baseline is dense UB for Δ_1 and UB+evict for Δ_2. Batch-invariant, deterministic kernels recover the coupling on the accelerator (§5.2). With default kernels the gap is measured as a share of changed draws and is not bounded analytically. We have not yet measured active KV memory blocks, wasted space between allocations, or maximum sustainable concurrent requests. Stronger sound prefix certificates and replication on optimized runtimes also remain future work.

Historical evidence before the clock repair. The component audit isolates memoization from state forking. The particle-scaling grid separates row savings from wall time.

Table 11: Verifier-store component audit: 30 fixed MBPP tasks, Event schedule (updates on each completed unit), $N = 8$, seed 31. Calls request prefix scores, executions run verifier code, and forks copy worker processes. Resident set size (RSS), the physical memory held by workers, sums worker residency and is an upper bound because unchanged pages are shared after a fork. MiB denotes mebibytes. Pass is visible-test success.

Interface	Pass (%)	Calls	Executions	Forks	Fork time (s)	RSS (MiB)
Memoization + state fork	63.3	87.7	34.2	20.6	0.041	80.5
Memoization only	63.3	87.7	41.9	20.6	0.040	31.5
No store	63.3	87.7	103.8	44.2	0.074	31.5

Table 12: Single-process particle scaling on 30 fixed MBPP tasks, full H100, three seeds. Because all measurements predate the clock repair, eager-arm differences may include a clock effect. Rows counts model-row evaluations, Verifier exec. counts executions, Wall is elapsed time, LM is language-model time, and Peak is peak accelerator memory. GiB denotes gibibytes. Pass is visible-test success. UB is unit barrier, cEPB is compact eager-prune barrier, and +evict removes every finished row.

N	Method	Pass (%)	Rows	Verifier exec.	Wall (s)	LM (s)	Peak (GiB)
32	UB	65.6	4891	80.3	3.4	2.4	3.72
32	UB+evict	66.7	1976	77.5	3.7	2.5	3.62
32	cEPB	65.6	3443	82.2	3.3	2.4	3.67
32	cEPB+evict	68.9	1901	82.2	3.1	2.3	3.62
64	UB	71.1	11838	163.5	5.5	3.4	4.10
64	UB+evict	71.1	3859	150.9	5.0	3.0	3.80
64	cEPB	70.0	7387	159.9	5.4	3.1	3.94
64	cEPB+evict	71.1	3837	159.7	5.5	3.1	3.81
128	UB	68.9	26544	324.8	8.6	5.2	4.89
128	UB+evict	71.1	7966	299.8	7.2	4.3	4.17
128	cEPB	72.2	16241	322.9	8.4	4.5	4.53
128	cEPB+evict	71.1	8114	323.4	8.3	4.5	4.19
Qwen2.5-Coder-3B							
64	UB	71.1	11755	171.4	6.4	4.4	7.37
64	UB+evict	73.3	5274	174.9	6.4	4.2	7.01
64	cEPB	75.6	8936	180.0	6.5	4.5	7.22
64	cEPB+evict	77.8	5362	179.4	6.4	4.4	7.01
DeepSeek-Coder-1.3B							
64	UB	71.1	11477	185.0	4.5	3.9	6.56
64	UB+evict	74.4	5724	187.9	4.0	3.4	5.03
64	cEPB	72.2	8741	187.9	4.8	3.7	5.82
64	cEPB+evict	72.2	5672	188.3	4.5	3.4	5.11

Pre-repair paired wall changes with 95% task-bootstrap intervals, retained as historical diagnostics that carry no current pruning-latency claim: $N = 32$, Δ_1: +0.4 s $[0.0, 0.9]$. $N = 32$, Δ_2: -0.6 s $[-1.5, -0.1]$. $N = 32$, total: -0.3 s $[-0.8, 0.1]$. $N = 64$, Δ_1: -0.5 s $[-0.9, -0.2]$. $N = 64$, Δ_2: +0.6 s $[0.1, 1.3]$. $N = 64$, total: 0.0 s $[-0.5, 0.6]$. $N = 128$, Δ_1: -1.4 s $[-2.6, -0.4]$. $N = 128$, Δ_2: +1.1 s $[0.1, 2.4]$. $N = 128$, total: -0.2 s $[-1.0, 0.5]$.

Eligibility of reported results. We exclude rows in which a cumulative soft potential was applied twice and MBPP rows that hard-pruned a visible-test failure before a later function redefinition could repair it. The latter rule removes no row from the studies reported here, whose cells are complete in every arm (Appendix A). Confirmatory aggregation crosses a fixed task subset with independent sampling seeds. Grids that change the task subset with the seed remain exploratory. Each full-cohort task, seed, and arm has one successful record. We also exclude diagnostics that treated a timeout or a repairable SQL failure as an absorbing zero. The two tasks that import `cmath` or `sympy` were evaluated under the corrected Python allowlist.

Additional contract-audit procedure. The two audit endpoints are implementation-independent: distance to an enumerated posterior and paired draws under shared streams. A runtime can be tested against them if it exposes a proposal stream indexed by logical slot, a shared barrier clock, and a per-row weight that survives physical removal. A fixed-schedule implementation claiming

equivalence must preserve the coupled clocks, weights, ancestors, and draws. An alternative schedule claiming consistency needs its own target argument and convergence evidence. Finite exact-target diagnostics can expose defects, but confidence-interval overlap is not an equivalence test and cannot establish consistency. The collapse clause is exercised only where collapse occurs (N=2 here). This procedure says nothing about serving throughput.

I CLAUSE-LEVEL ZEROS: THE BIRD MINI-DEV STUDY

The SQL verifier scores completed clauses against the question's read-only SQLite database. We use `sqlglot` to recognize clause keywords outside nested expressions. We execute a prefix only after its FROM clause closes. Execution is capped at one row and five seconds. We certify zero only when the fixed leading text of an error message names an unknown table, function, module, collation (text-comparison rule), pragma (database-setting directive), or database. The model cannot forge this leading text by choosing an identifier. Other failures, including timeouts and unrecognized diagnostics, are neutral. This is a conservative rule for one engine and no proof across SQL dialects.

The matched study uses 500 Mini-Dev questions in source order, the pinned model revision, $N = 8$, temperature $\tau = 1$, and seeds 0, 1, 2. The four arms share the unit-step clock. Table 4 reports the resulting Δ_2 contrast. An interval covering zero leaves the incremental saving unresolved. Execution accuracy compares predicted and reference result sets using the retained execution-equivalence evaluator. Scoring is distinct from generation-time verification.

J EARLY CERTIFIED ZEROS: THE ARITHMETIC PREDICATE

A completed `<<expression=value>>` calculator annotation is append-stable. Our implemented check, the operational predicate, evaluates integers, decimals, $+$, $-$, $*$, $/$, the remainder operator $\%$, and parentheses in IEEE 754 double precision, the standard 64-bit floating-point representation, with relative and absolute tolerance 10^{-6}. A mismatch yields zero, a match yields one, and syntax outside the supported grammar yields the neutral factor. Unsupported does not mean false. The terminal constraint H requires no rejected annotation and a final `#### <number>` line. The gold answer enters evaluation only.

The predicate is not an exact-arithmetic oracle. For example, $10^{16} + 1 - 10^{16} = 1$ is rejected by floating-point evaluation but accepted by the rational reference. The reference parses decimal literals directly from source and evaluates only the operator present in the expression. Regression tests cover zero operands and long decimals. Our all-cell certificate replay retains each pruned annotation, prefix hash, first-zero unit and remaining token budget. Appendix B reports both strict rational equality and tolerance-matched results, which must not be conflated.

The exploratory arithmetic domain was selected after examining code and SQL results. The subsequent fresh-problem study and paired harder variants test the declared mechanism: certified-zero opportunities determine the potential for early removal, and answer accuracy alone does not.

Table 13: Eager-pruning regime map at $N = 64$ (20,000 trials/cell). Here p_0 is the probability that a fast first unit yields zero, and L is the number of tokens valid particles need to reach the barrier. Values are logical-token reduction over UB, computed from retained cell means.

p_0	$L = 2$	$L = 4$	$L = 8$
0.1	2.6%	4.0%	4.6%
0.3	8.8%	13.3%	15.4%
0.5	16.6%	25.0%	29.2%
0.7	27.0%	40.3%	47.1%

Certificate-frequency regimes. Table 13 maps the logical-token reduction of eager pruning over UB on the enumerable target to the probability p_0 that a fast first unit yields a certified zero and to the number L of tokens valid particles need to reach the barrier. It conditions on certificate opportunities and says nothing about model quality. For an illustrative comparison, the exploratory BIRD rate of 0.38 certified zeros per run of $N = 8$ particles is about 0.05 certificates per particle, which lies

below the $p_0 = 0.1$ row (2.6 to 4.6% fewer logical tokens), and the GSM-Hard rate of 4.36 per run is about 0.5 per particle, near the $p_0 = 0.5$ row (16.6 to 29.2%). This reading assumes that certificates arrive on the first unit at that per-particle rate and that the target's unit structure transfers, and neither assumption is measured for any model beyond the one we ran.