Reliable RL Scaling Requires Accounting for Prefill–Decode Kernel Mismatch

Yifan Zhang et al.

yifanzhangresearch@gmail.com

August 9, 2026

Abstract

A language-model rollout is on-policy for a declared target only when the post-sampling-transform behavior distribution equals that target distribution on the histories reached by rollout. Equal checkpoints do not ensure this condition. Prompt prefill, cached decoding, teacher-forced scoring, parallel scans, recurrent state updates, cache precision, quantization, and reduction order can all change token probabilities. For linear attention and state-space models, a small state discrepancy can propagate through all later updates. Moreover, forward probability agreement at one parameter value is not enough to claim optimization of a recurrent deployment policy: the learner must also use the corresponding score function, or a backward-equivalent implementation.

We study four responses. First, treat execution mismatch as off-policy data, store the actor-emitted probability, and use an explicit importance ratio. We distinguish the exact local one-step-deviation surrogate from full trajectory correction and state the required support contract. Second, make one chunkwise or recurrent execution rule canonical across rollout and differentiable learner replay. TTT, Titans, and RWKV-7 illustrate relevant execution semantics, but architecture names alone do not establish parity. Third, use higher precision for recurrent states and sensitive updates as a numerical mitigation, not as an equality proof. Fourth, use a fast path only as a proposal and let a recurrent target perform exact modified rejection sampling; the standard tokenwise correction is exact only when the proposal exposes the actual conditional probabilities of the drafted sequence. We recommend separating an exact recurrent-target mode from a scalable parallel-target surrogate, with explicit audits for forward probabilities, score functions, support, and state precision.

Project Page: https://github.com/yifanzhang-pro/Pretraining-RL-Science

1 Introduction

A model checkpoint does not by itself determine the policy executed by a language-model system. The executable path also includes attention or recurrence kernels, prompt-state construction, cache layout, precision, quantization, parallel layout, reduction order, masks, and the sampling transform. Two engines with identical parameters can therefore define different token distributions.

Let K denote the complete execution operator relevant to one token decision. If $z_\theta^{\mathsf{K}}(h)$ is the logit vector at history h, define the effective policy

$$\pi_\theta^{\mathsf{K}}(\cdot \mid h) = \mathcal{T}_{\mathsf{K}}\left(z_\theta^{\mathsf{K}}(h)\right), \tag{1.1}$$

where $\mathcal{T}_{\mathsf{K}}$ includes temperature, admissibility masks, truncation, and any probability floor used by that path. A rollout record is distributionally on-policy for the learner-defined target at parameters θ only if

$$\pi_\theta^{\mathsf{K}_{\mathrm{roll}}}(\cdot \mid h) = \pi_\theta^{\mathsf{K}_{\mathrm{learn}}}(\cdot \mid h) \tag{1.2}$$

for every history reached with positive rollout probability. Equal weights, checkpoint identifiers, token sequences, or model names do not imply Eq. (1.2).

There is a second contract. Define the score of an executable policy by

$$s_\theta^{\mathsf{K}}(a, h) := \nabla_\theta \log \pi_\theta^{\mathsf{K}}(a \mid h). \tag{1.3}$$

Forward equality in Eq. (1.2) is enough to make the data on-policy for the learner distribution at that fixed θ. It is not, by itself, enough to show that a parallel learner computes the gradient of a recurrent deployment policy. If the scientific target is $\pi_\theta^{\mathsf{K}_{\mathrm{deploy}}}$, the learner must use $s_\theta^{\mathsf{K}_{\mathrm{deploy}}}$—equivalently, differentiate $\log \pi_\theta^{\mathsf{K}_{\mathrm{deploy}}}$—or use an implementation proved equivalent in both forward and backward computations. Equality of two forward distributions at one parameter value does not imply equality of their Jacobians.

The common implementation error is to sample with $\mathsf{K}_{\mathrm{roll}}$ but later reconstruct the "old" probability with $\mathsf{K}_{\mathrm{learn}}$. For a learner-defined target, the correct token-local likelihood ratio is

$$\rho_t = \frac{\pi_{\theta_n}^{\mathsf{K}_{\mathrm{learn}}}(a_t \mid h_t)}{\pi_{\theta_v}^{\mathsf{K}_{\mathrm{roll}}}(a_t \mid h_t)}, \tag{1.4}$$

where v is the actor version and n is the learner version. The buggy recomputation instead uses

$$\widetilde{\rho}_t = \frac{\pi_{\theta_n}^{\mathsf{K}_{\mathrm{learn}}}(a_t \mid h_t)}{\pi_{\theta_v}^{\mathsf{K}_{\mathrm{learn}}}(a_t \mid h_t)}. \tag{1.5}$$

At zero parameter lag, $n = v$, Eq. (1.5) is one by construction, while Eq. (1.4) can still differ from one because the actor and learner executed different operators. Learner-side recomputation therefore hides the kernel gap rather than correcting it. The ratio in Eq. (1.4) exactly changes the conditional action law at a behavior history; it is not, by itself, a full trajectory-distribution correction. We make that distinction precise in Sec. 3.

2 Executable Policy Mismatch

2.1 Softmax Attention: Prefill and Decode

A standard autoregressive rollout has two phases. Prompt prefill constructs the KV cache and normally produces the distribution for the first generated token. Each later response token is produced by a one-token decode step whose state depends on that prefill cache and all preceding sampled tokens. The learner usually concatenates the prompt and sampled response and evaluates them in one teacher-forced causal pass. Thus token 1 compares prompt-only prefill with a full-pass causal score, while tokens $t \geq 2$ compare a composed prefill–decode trace with that parallel pass. Either comparison can differ in finite precision, as shown in Fig. 1.

In exact real arithmetic, parallel causal softmax attention and the corresponding KV-cached recurrence can represent the same mathematical function. That algebraic identity is not a finite-precision equality certificate. Implementations can differ in

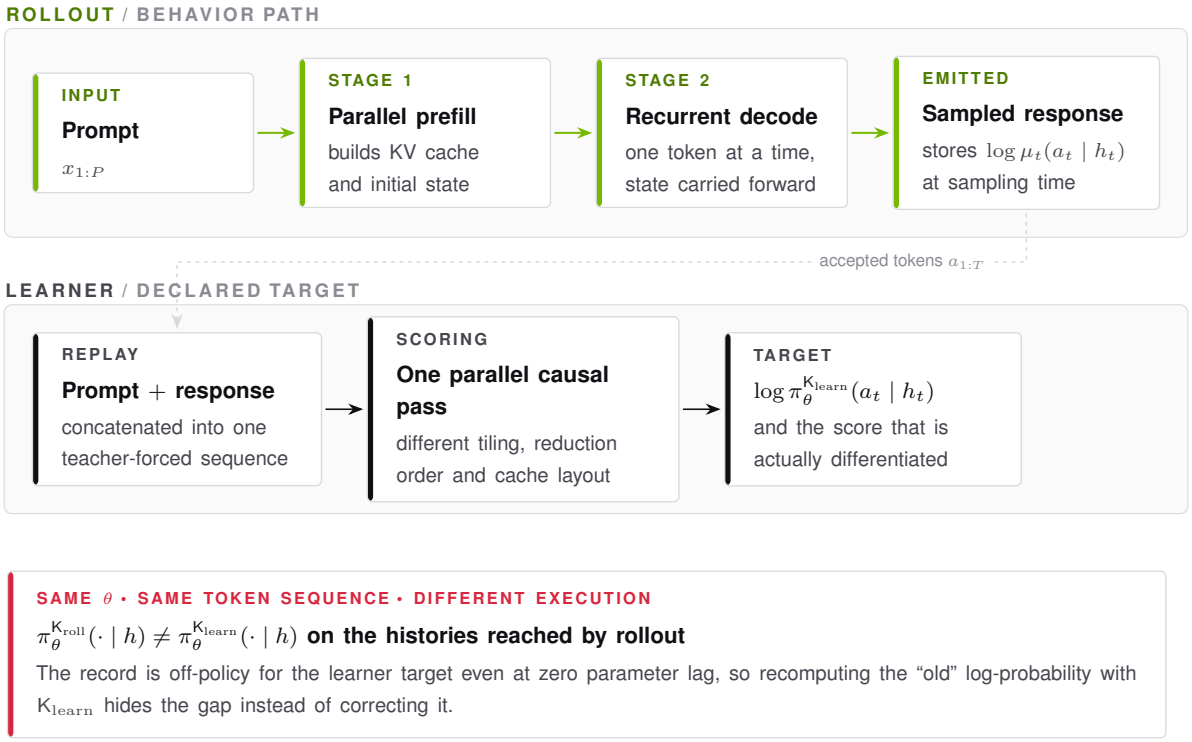

Figure 1 A common actor–learner split. The behavior path is a composed trace: prompt prefill constructs the initial cache or state, and recurrent decode extends it token by token. The learner usually scores the same response with a single parallel teacher-forced pass, so the two paths define different executable policies at identical parameters.

- tiling, fusion, and accumulation order;
- KV-cache layout and prompt-state construction;
- batch shape and sequence length;
- tensor- and sequence-parallel communication;
- precision, quantization, and reduction trees; and
- the post-logit sampling transform and support mask.

Any of these can perturb logits or change a truncation boundary. The resulting gap may be negligible in one configuration and large enough to destabilize RL in another, so it must be measured under the production configuration rather than inferred from checkpoint identity (Zhong et al., 2026; Zhang et al., 2025).

The sampled-token log-ratio can be decomposed into a fixed-weight execution gap and an actor-

version gap:

$$\log \rho_{t,n,v} = \underbrace{\log \pi_{\theta_n}^{\mathsf{K}_{\text{learn}}}(a_t \mid h_t) - \log \pi_{\theta_n}^{\mathsf{K}_{\text{roll}}}(a_t \mid h_t)}_{\Delta_{t,n}^{\text{kernel}}} + \underbrace{\log \pi_{\theta_n}^{\mathsf{K}_{\text{roll}}}(a_t \mid h_t) - \log \pi_{\theta_v}^{\mathsf{K}_{\text{roll}}}(a_t \mid h_t)}_{\Delta_{t,n,v}^{\text{stale}}}. \tag{2.1}$$

When $n = v$, the staleness term vanishes but the execution term need not. A fully synchronous pipeline can therefore still be off-policy for the learner-defined target. Computing the two terms separately requires replaying the rollout operator at θ_n; only their sum is directly available from the stored actor log-probability and the learner score.

What contracts are sufficient? The strongest route is differentiable replay through the same prompt-state construction, KV-cache update, precision, masks, and tokenwise execution used by rollout. The learner must reconstruct the cache under its own parameters and propagate gradients through it; reusing a detached actor cache omits parameter dependence through earlier states. A parallel path is also valid as the learner-defined target. It is literally on-policy at fixed weights only when its forward distribution equals rollout; agreement within a stated tolerance supports only an approximate claim. To claim that the parallel path computes the gradient of recurrent deployment, one additionally needs backward or score-function equivalence, not only forward probability parity.

2.2 Linear Attention and State-Space Models

For linear attention and state-space models, the distinction is structural. A tokenwise implementation is naturally a state transition

$$(o_t, S_t) = \Phi_\theta(S_{t-1}, x_t), \tag{2.2}$$

whereas high-throughput training often uses a parallel scan, a convolutional dual, or a chunkwise factorization.

In common chunkwise formulations, the chunk size C spans two schedule endpoints:

$$C = 1 \iff \text{tokenwise recurrence}, \qquad C = L \iff \text{one full-sequence chunk}. \tag{2.3}$$

The $C = L$ endpoint may compute causal outputs in parallel inside the chunk; the exact schedule is architecture- and kernel-dependent. Intermediate chunk sizes trade sequential state propagation for matrix-multiplication throughput (Yang et al., 2024a,b). Even when recurrent, scan, and chunkwise formulas are algebraically equivalent, changing the scan tree, chunk boundaries, state materialization points, or accumulation precision changes finite-precision parenthesization. A discrepancy at one boundary can affect every later state and token probability.

A recurrent actor paired with a chunkwise learner must therefore be treated as two effective policies unless parity has been established for the exact chunk schedule, boundary convention, precision, parallel layout, masks, and prompt-state construction used in the run.

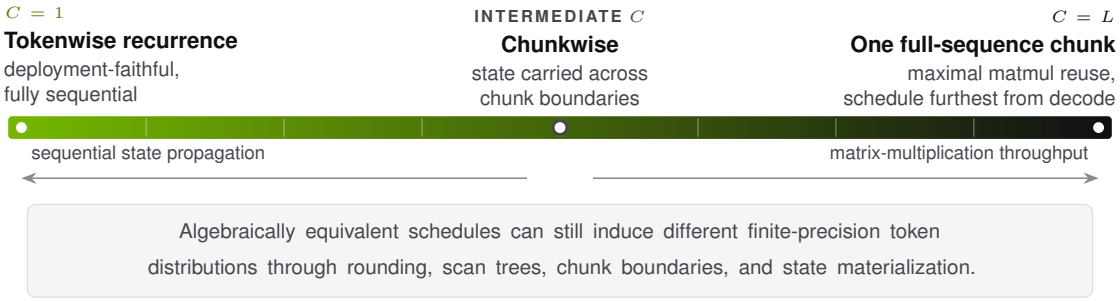

Figure 2 Chunk size selects an execution schedule, trading sequential state propagation for matrix-multiplication throughput. Algebraic equivalence does not by itself establish equality of the induced finite-precision token distributions.

2.3 DeltaNet as a Sensitive Case

DeltaNet (Schlag et al., 2021) makes schedule-dependent state error especially consequential because the state update is a data-dependent corrective write. In one common orientation,

$$S_t = S_{t-1}\left(I - \beta_t k_t k_t^\top\right) + \beta_t v_t k_t^\top, \tag{2.4}$$

which is equivalent to $S_t = S_{t-1} + \beta_t(v_t - S_{t-1}k_t)k_t^\top$. Common high-throughput training implementations rewrite the ordered recurrence using products of Householder-like factors and a memory-efficient chunkwise representation (Yang et al., 2024b). The factors generally do not commute, so temporal order must be preserved algebraically. Even when it is preserved, the factorization changes floating-point parenthesization and the points at which intermediate values are rounded.

The discrepancy then enters future updates. An error in S_{t-1} changes the read $S_{t-1}k_t$, the erase residual $v_t - S_{t-1}k_t$, and hence the next state. State error is therefore not merely observed downstream; it participates in subsequent writes. Recurrent DeltaNet rollout and learner scoring with a parallelized delta-rule kernel should be regarded as different effective policies unless forward and, when required, backward parity are demonstrated under the production configuration. The same warning applies to gated delta-rule variants and other recurrent architectures whose training kernel changes the update schedule.

3 Off-Policy RL under Execution Mismatch

The safe systems abstraction is

$$\text{effective policy} = \text{parameters} + \text{execution trace} + \text{sampling transform}. \tag{3.1}$$

Two contracts must then be declared separately: the *behavior-distribution contract*, which identifies the probability that generated each token, and the *target-score contract*, which identifies the executable policy whose log-probability is differentiated.

This yields three implementation rules.

1. **Capture post-transform behavior probabilities from the actor path.** Store the log-probability emitted by the actual sampling engine when each token is drawn. Raw model logits or a learner-side replay are not substitutes unless the full sampling transform and distributional parity are established.

2. **Separate kernel mismatch from parameter staleness.** The total learner-to-actor log-ratio is available on every record. Its kernel and version components require an additional current-weight rollout replay and should be measured on a controlled audit subset.

3. **Reserve exact claims for the corresponding contract.** Forward equality makes data on-policy for the declared learner distribution. Exact optimization of a recurrent deployment policy additionally requires the recurrent score or a forward-and-backward equivalent implementation.

When the forward distributions differ, off-policy is the correct abstraction even at zero actor lag. Replacing the actor denominator with a learner-engine recomputation discards the actual behavior law and breaks the intended change of measure. A fixed-weight audit for both probabilities and score functions is given in Sec. A.1.

The most general response is to preserve the distribution that actually generated each token and perform an explicit change of measure at the learner. Throughout this section, $\mathsf{K}_{\mathrm{learn}}$ denotes the *declared differentiable target operator*, not merely whichever kernel is convenient. If the target is recurrent deployment, then $\mathsf{K}_{\mathrm{learn}}$ must implement $\mathsf{K}_{\mathrm{recur}}$ in both forward and backward computations, or be proved equivalent to it.

3.1 A time-dependent behavior trace

A rollout does not use one isolated kernel for the entire response. For a softmax-attention actor, a useful shorthand is

$$\mathsf{K}_{\mathrm{beh},t} = \begin{cases} \mathsf{K}_{\mathrm{prefill}}, & t = 1, \\ \mathsf{K}_{\mathrm{decode}}^{(t-1)} \circ \mathsf{K}_{\mathrm{prefill}}, & t \geq 2, \end{cases} \tag{3.2}$$

where $\mathsf{K}_{\mathrm{decode}}^{(t-1)}$ denotes the stateful trace through the preceding response tokens $a_{1:t-1}$. Thus the label "decode" includes the prompt state created by prefill and all earlier decode transitions. For a linear-attention or state-space actor, $\mathsf{K}_{\mathrm{beh},t}$ similarly denotes the complete recurrent, scan, or chunkwise trace that produced the state used at token t.

Let

$$\mu_{v,t}(\cdot \mid h_t) := \pi_{\theta_v}^{\mathsf{K}_{\mathrm{beh},t}}(\cdot \mid h_t) \tag{3.3}$$

be the post-transform behavior distribution at actor version v, and let

$$p_{n,t}(\cdot \mid h_t) := \pi_{\theta_n}^{\mathsf{K}_{\mathrm{learn}}}(\cdot \mid h_t) \tag{3.4}$$

be the learner-defined target. The token-local ratio is

$$\rho_{t,n,v} = \frac{p_{n,t}(a_t \mid h_t)}{\mu_{v,t}(a_t \mid h_t)} = \exp\left(\ell_{t,n}^{p} - \ell_{t,v}^{\mu}\right), \tag{3.5}$$

where $\ell_{t,v}^{\mu}$ is captured from the actor path at sampling time and never overwritten. Recomputing it with $\mathsf{K}_{\mathrm{learn}}$ changes the denominator to a different policy.

3.2 Prefill–decode and chunkwise–recurrent gaps are one change of measure

At fixed learner parameters, the kernel factor can be decomposed through bridge operators $\mathsf{K}_0, \ldots, \mathsf{K}_M$, with $\mathsf{K}_0 = \mathsf{K}_{\mathrm{beh},t}$ and $\mathsf{K}_M = \mathsf{K}_{\mathrm{learn}}$:

$$\frac{\pi_{\theta_n}^{\mathsf{K}_{\mathrm{learn}}}(a_t \mid h_t)}{\pi_{\theta_n}^{\mathsf{K}_{\mathrm{beh},t}}(a_t \mid h_t)} = \prod_{j=1}^{M} \frac{\pi_{\theta_n}^{\mathsf{K}_j}(a_t \mid h_t)}{\pi_{\theta_n}^{\mathsf{K}_{j-1}}(a_t \mid h_t)}. \tag{3.6}$$

For softmax attention, a bridge may compare teacher-forced scoring with the composed prefill–decode trace. For a state-space layer, another may compare chunkwise and tokenwise execution. These factors are useful diagnostics only if they share exact intermediate distributions and telescope to Eq. (3.5). Multiplying independently defined "corrections" double-counts the mismatch.

Separating fixed-weight execution and actor staleness gives

$$\rho_{t,n,v} = \underbrace{\frac{\pi_{\theta_n}^{\mathsf{K}_{\mathrm{learn}}}(a_t \mid h_t)}{\pi_{\theta_n}^{\mathsf{K}_{\mathrm{beh},t}}(a_t \mid h_t)}}_{\rho_{t,n}^{\mathrm{kernel}}} \underbrace{\frac{\pi_{\theta_n}^{\mathsf{K}_{\mathrm{beh},t}}(a_t \mid h_t)}{\pi_{\theta_v}^{\mathsf{K}_{\mathrm{beh},t}}(a_t \mid h_t)}}_{\rho_{t,n,v}^{\mathrm{stale}}}. \tag{3.7}$$

The first term can remain at zero parameter lag; the second can remain when actor and learner use the same kernel but different versions.

3.3 What a token ratio corrects

Consider a finite-horizon process $\tau = (h_1, a_1, r_1, \ldots, h_H, a_H, r_H)$ with parameter-independent environment dynamics and rewards, and assume the declared target probabilities are differentiable at the parameter value being updated. Let

$$G_t(\tau) = \sum_{u=t}^{H} r_u, \qquad J_{\mathsf{K}_{\mathrm{learn}}}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta^{\mathsf{K}_{\mathrm{learn}}}} \left[\sum_{t=1}^{H} r_t \right]. \tag{3.8}$$

Let d_μ^t be the behavior history distribution and

$$Q_t^\mu(h, a) = \mathbb{E}_{\tau \sim \mu}[G_t \mid h_t = h, a_t = a]. \tag{3.9}$$

Holding the behavior occupancy and continuation fixed, define the one-step-deviation surrogate

$$\mathcal{S}_\mu(\theta) = \sum_{t=1}^{H} \mathbb{E}_{h \sim d_\mu^t} \mathbb{E}_{a \sim \pi_\theta^{\mathsf{K}_{\mathrm{learn}}}(\cdot \mid h)}[Q_t^\mu(h, a)]. \tag{3.10}$$

Under the support condition

$$\pi_\theta^{\mathsf{K}_{\mathrm{learn}}}(\cdot \mid h) \ll \mu_t(\cdot \mid h) \tag{3.11}$$

for every admitted behavior history,

$$\nabla_\theta \mathcal{S}_\mu(\theta) = \mathbb{E}_{\tau \sim \mu} \left[\sum_{t=1}^{H} \rho_t G_t \nabla_\theta \log \pi_\theta^{\mathsf{K}_{\mathrm{learn}}}(a_t \mid h_t) \right]. \tag{3.12}$$

An action-independent baseline $b_t(h_t)$ may replace G_t by $G_t - b_t(h_t)$. Equation (3.12) exactly corrects the conditional action law at behavior histories. It leaves the history distribution d_μ^t and behavior continuation inside Q_t^μ unchanged.

Importance weighting changes the sampling measure; it does not replace the score Jacobian. If the intended target is the recurrent deployment policy, then both the numerator and score must be recurrent:

$$\rho_t^{\mathrm{recur}} = \frac{\pi_\theta^{\mathsf{K}_{\mathrm{recur}}}(a_t \mid h_t)}{\mu_t(a_t \mid h_t)}, \qquad s_t^{\mathrm{recur}} = \nabla_\theta \log \pi_\theta^{\mathsf{K}_{\mathrm{recur}}}(a_t \mid h_t). \tag{3.13}$$

A recurrent ratio multiplied by a parallel-kernel score is generally not the gradient of either executable policy. If recurrent differentiation is unavailable, the update must be described as a cross-operator surrogate rather than an exact recurrent-policy gradient.

3.4 Local correction versus full trajectory correction

Let

$$W_t = \prod_{u=1}^{t} \rho_u, \qquad W_0 = 1. \tag{3.14}$$

Assume target trajectories are absolutely continuous with respect to behavior trajectories. Then W_{t-1} corrects the history law up to h_t, and ρ_t corrects the current action. A return sampled under behavior still contains behavior-policy continuation, so multiplying G_t only by W_t is not generally a complete target-policy correction. With the convention that r_u is measurable after action a_u, an exact per-decision identity is

$$\nabla J_{\mathsf{K}_{\mathrm{learn}}}(\theta) = \mathbb{E}_{\tau\sim\mu}\left[\sum_{u=1}^{H} W_u r_u \sum_{t=1}^{u} \nabla_\theta \log \pi_\theta^{\mathsf{K}_{\mathrm{learn}}}(a_t \mid h_t)\right]. \tag{3.15}$$

For a terminal reward $R(\tau)$ this reduces to

$$\nabla J_{\mathsf{K}_{\mathrm{learn}}}(\theta) = \mathbb{E}_{\tau\sim\mu}\left[W_H R(\tau) \sum_{t=1}^{H} \nabla_\theta \log \pi_\theta^{\mathsf{K}_{\mathrm{learn}}}(a_t \mid h_t)\right]. \tag{3.16}$$

The variance of W_t or W_H can be prohibitive. Token-local weighting is therefore a deliberate local surrogate, not a claim of full trajectory correction.

A practical token-normalized score loss is

$$\begin{aligned} N_m &= \sum_i w_i \sum_t m_{i,t}, \\ \mathcal{L}_{\mathrm{IS}}(\theta) &= -\frac{1}{N_m} \sum_i w_i \sum_t m_{i,t}\, \mathrm{stopgrad}(\rho_{i,t} A_{i,t}) \log \pi_\theta^{\mathsf{K}_{\mathrm{learn}}}(a_{i,t} \mid h_{i,t}), \end{aligned} \tag{3.17}$$

where $m_{i,t}$ selects policy tokens, $w_i \geq 0$ is a fixed record weight, $N_m > 0$, and $A_{i,t}$ is a return or a valid action-independent-baseline advantage. Per-sequence normalization is also possible, but it defines a different length weighting and should be stated explicitly. The behavior log-probability inside $\rho_{i,t}$ remains immutable across learner steps.

3.5 Support, variance, and finite-step control

Let $\mathcal{A}_{\mathrm{adm}}(h)$ be the action set allowed by shared, parameter-independent constraints such as a grammar or safety mask. The token-local identity requires support domination on this declared set at every admitted behavior history; the trajectory identities additionally require domination along every target prefix. Hard top-k, top-p, min-p, or other logit-dependent truncation can nevertheless set $\mu(a \mid h) = 0$ while the learner target assigns positive mass, especially when kernels change the cutoff ordering. No finite ratio can recover those missing actions.

In exact arithmetic, an untruncated softmax on $\mathcal{A}_{\mathrm{adm}}(h)$ has full support, but an executable implementation must still verify that finite-precision sampling and storage do not round admissible probabilities to zero. A more explicit option is a shared probability-floor transform

$$\mathcal{F}_{T,\varepsilon}(z)(a) = (1-\varepsilon)\frac{\exp(z_a/T)}{\sum_{b\in\mathcal{A}_{\mathrm{adm}}(h)} \exp(z_b/T)} + \varepsilon u_h(a), \tag{3.18}$$

where $T > 0$, $u_h \in \Delta(\mathcal{A}_{\mathrm{adm}}(h))$, $u_h(a) > 0$ for every $a \in \mathcal{A}_{\mathrm{adm}}(h)$, and both terms are zero outside $\mathcal{A}_{\mathrm{adm}}(h)$. The same transform must be used for actor sampling, stored probabilities, learner scoring, differentiation, and divergence measurement. A hard-truncated run can instead define a target restricted to the stored actor support, but that is a support-restricted surrogate rather than the original untruncated target.

Clipping or truncating ρ is an explicit bias–variance tradeoff. It cannot repair missing support and must not be presented as exact correction. Exact mode uses the raw finite ratio; a stabilized mode may use a declared bounded transformation $\bar{\rho} = \psi(\rho)$ or self-normalization, while retaining the raw log-ratio for diagnostics. A clipped surrogate may also be used as an approximate trust-region device, but it is then no longer an exact change-of-measure estimator.

At a fixed history,

$$\mathbb{E}_{a\sim\mu}[\rho(a)^2] = 1 + \chi^2\left(\pi_\theta^{\mathsf{K}_{\mathrm{learn}}} \middle\| \mu\right), \qquad \chi^2(p\|m) = \sum_a \frac{\big(p(a) - m(a)\big)^2}{m(a)}. \tag{3.19}$$

A production implementation should report raw log-ratio quantiles, second moments, effective sample size, support violations, and nonfinite values. Exact per-history KL or TV requires full behavior distributions or reproducible replay of the actor checkpoint and operator; sampled action log-ratios alone provide only noisy, direction-specific estimates.

4 Aligning the Executable Policy

Importance weighting accepts that actor and learner define different policies. The alternative is to make one recurrence or chunk rule part of the model semantics and execute it consistently.

4.1 Canonical chunkwise updates

Let the accepted token stream be divided into blocks $X_j = x_{jC+1:(j+1)C}$. A semantic chunkwise state machine can be written as

$$O_j = G_\theta^{(C)}(S_j, X_j), \qquad S_{j+1} = F_\theta^{(C)}(S_j, X_j). \tag{4.1}$$

Literal alignment requires actor and learner to share the same C, boundary convention, within-chunk causal mask, read-before-write rule, optimizer-like memory update, precision, and state-materialization points.

TTT layers define the hidden state as a model updated by a self-supervised learning step and use mini-batches of tokens plus a dual form for hardware efficiency (Sun et al., 2024). Titans likewise reformulates mini-batch memory updates with matmuls and parallel scans, and some variants segment the sequence explicitly (Behrouz et al., 2025). These works show that chunking can be part of model semantics rather than a hidden implementation detail. They do not imply that any two kernels called "TTT", "Titans", or "chunkwise" are equal. Changing chunk size, momentum carry, boundary resets, update timing, or scan tree defines a different executable policy unless equivalence is established.

A chunkwise memory update may process a block of *already accepted* tokens together. It does not make unknown stochastic tokens available in advance. Exact parallel token generation still requires a proposal-and-verification mechanism, discussed in Sec. 4.4.

EXECUTION CHOICE	REQUIRED CONTRACT	MAIN BENEFIT	MAIN COST / CAVEAT
Parallel target, recurrent actor	Store actor probability; use explicit IS	Highest learner throughput	Exact only for the declared surrogate or with trajectory products
Canonical chunk rule	Same semantics, boundaries, precision, and backward	Chunk-level matmul efficiency	Unknown autoregressive tokens remain sequential
Pure recurrent replay	Same tokenwise path with gradients through state	Strongest deployment-gradient contract	Serial sequence replay and activation cost
Verified parallel implementation	Forward parity for policy; backward parity for deployment gradient	Parallelism without a fixed-weight kernel correction	Verification is configuration-specific

Table 1 Execution alignment trades throughput flexibility for stronger forward- and backward-equivalence contracts.

4.2 Pure recurrent training and inference

The strongest execution contract uses the same tokenwise recurrence everywhere:

$$S_t = F_\theta(S_{t-1}, x_t), \qquad z_t = G_\theta(S_t, x_t), \qquad a_t \sim \mathcal{T}(z_t). \tag{4.2}$$

Rollout, differentiable learner replay, validation, and deployment execute Eq. (4.2) with the same state initialization, order, precision, and masks. The learner reconstructs states under its own parameters and differentiates through them.

RWKV-7 is a useful architectural example because it provides constant-memory, constant-time recurrent inference per token while retaining parallelizable training (Peng et al., 2025). Therefore, "use RWKV-7" is not a parity certificate. The learner must intentionally differentiate the recurrent form, or use a parallel implementation shown equivalent in both forward and backward computations.

Tokenwise replay sacrifices sequence-dimension parallelism and can be expensive for long responses. Activation checkpointing or exact recomputation can reduce memory without changing the mathematical gradient, provided the recomputation is deterministic under the declared numerical contract. Truncated backpropagation and state detachment do change the gradient contract. The benefit of recurrent replay is conceptual clarity: at fixed weights, there is no parallel–recurrent importance factor when the same recurrence defines both behavior and target.

4.3 Higher Precision for Recurrent State

Linear-attention and state-space layers feed a stored state into every subsequent update. To isolate numerical effects, consider a fixed teacher-forced input sequence and let S_t be a reference recurrence, $\widehat{S}_t$ an implementation, and $e_t = \widehat{S}_t - S_t$. A generic perturbation bound is

$$\|e_t\| \le L_t \|e_{t-1}\| + \delta_t^{\text{round}} + \delta_t^{\text{schedule}}, \tag{4.3}$$

where L_t is a local state-sensitivity factor, δ_t^{round} is arithmetic error under a fixed schedule, and $\delta_t^{\text{schedule}}$ captures a changed scan tree, chunk boundary, or materialization rule. Unrolling gives

$$\begin{aligned}\|e_t\| \le & \left(\prod_{j=1}^{t} L_j\right) \|e_0\| \\ & + \sum_{u=1}^{t} \left(\prod_{j=u+1}^{t} L_j\right) \left(\delta_u^{\text{round}} + \delta_u^{\text{schedule}}\right).\end{aligned} \tag{4.4}$$

Higher precision can reduce δ^{round}. It does not remove schedule error, nor does it prevent amplification when products of the L_j are large.

A correctness-first precision ladder is:

1. **Establish an FP32 state baseline.** Keep recurrent states, decay products, normalization accumulators that feed the recurrence, and state-update reductions in FP32 while allowing unrelated embeddings, MLPs, and communication to remain in BF16 or FP16.

2. **Promote sensitive parameters and operands as needed.** Recurrence-defining gates, time constants, and update operands may also require FP32. The official Mamba implementation notes that storing main parameters in higher precision can be necessary because SSMs are sensitive to recurrent dynamics (Gu and Dao, 2023; State Spaces Team, 2026).

3. **Use FP64 as a reference, not a default training format.** FP64 recurrent states and updates are useful for small-model audits, sampled boundary checks, and diagnosing whether remaining error is numerical or semantic.

For DeltaNet-like updates, a useful minimum is to compute $S_{t-1}k_t$, the residual $v_t - S_{t-1}k_t$, and the rank-one accumulation in FP32. If discrepancies remain, promote the recurrent state and recurrence-defining operands to FP64 in the audit path. Casting an already rounded BF16 state to FP32 cannot recover lost information; the higher precision must be used before and during accumulation.

Higher precision is a mitigation, not an on-policy certificate. Two FP64 programs with different scan trees can still disagree, and hard truncation can turn a small logit perturbation into a support change. Precision experiments should therefore be paired with the probability, state, and score audits in Sec. A.1.

4.4 Fast Proposal with Recurrent Verification

A fourth route separates speed from policy semantics. Let the recurrent implementation define the canonical post-transform target distribution

$$p_i(\cdot) = \pi_\theta^{\mathsf{K}_{\text{recur}}}(\cdot \mid h_i), \tag{4.5}$$

and let a faster operator define proposal conditionals q_i. The standard tokenwise modified-rejection algorithm is exact when the drafted block is actually sampled from

$$q(y_{1:\gamma} \mid h_1) = \prod_{i=1}^{\gamma} q_i(y_i \mid h_1, y_{<i}), \tag{4.6}$$

Algorithm 1 Conditional proposal with recurrent target verification.

Require: Committed recurrent state S; proposal conditionals q_i; recurrent target $\mathsf{K}_{\mathrm{recur}}$; draft length γ.

Sample $y_{1:\gamma}$ from the proposal law in Eq. (4.6); retain every $q_i(\cdot \mid h_1, y_{<i})$.
From a temporary copy of S, run the recurrent target along the candidate prefix to obtain $p_1, \dots, p_\gamma$ and temporary target states.
for $i = 1, \dots, \gamma$ **do**
    Assert $q_i(y_i) > 0$ and draw $u_i \sim \mathrm{Uniform}(0, 1)$.
    if $u_i \le \min(1, p_i(y_i)/q_i(y_i))$ **then**
        Commit y_i and the corresponding recurrent target state transition.
    else
        Draw $z \sim \mathrm{Normalize}([p_i - q_i]_+)$ from the state before position i.
        Commit z through the recurrent target; discard all later proposal and temporary states; **stop**.
    end if
end for
If every draft token is accepted, optionally sample one additional token from the next recurrent target distribution.

and the recorded q_i are those conditional distributions. The target verifier evaluates $p_i(\cdot \mid h_1, y_{<i})$ on the same candidate prefix.

For each proposed token, accept with probability

$$\alpha_i(y_i) = \min\left(1, \frac{p_i(y_i)}{q_i(y_i)}\right). \tag{4.7}$$

If token y_i is rejected, draw a replacement from

$$r_i(a) = \frac{\big[p_i(a) - q_i(a)\big]_+}{\sum_b \big[p_i(b) - q_i(b)\big]_+}. \tag{4.8}$$

Applied from left to right, this recovers the recurrent target distribution up to the declared hardware numerics (Leviathan et al., 2023; Chen et al., 2023). Argmax agreement, a logit tolerance, or arbitrary accept-then-resample logic does not generally preserve stochastic target sampling.

A teacher-forced prefill kernel can score a supplied candidate block, but it cannot by itself generate later autoregressive tokens because each position depends on the candidate prefix. A valid drafter may be a smaller autoregressive model, a same-model approximate recurrence, or another proposal mechanism whose actual conditional or joint law is available. For an arbitrary non-autoregressive or correlated block proposal, marginal token probabilities are not enough: one needs a joint/block correction or a specialized proof. Applying Eqs. (4.7) and (4.8) to convenient marginals is only a heuristic.

This arrangement reverses the usual performance emphasis. Standard speculative decoding often uses a cheap autoregressive drafter and a parallel target verifier. Here the recurrent implementation is the correctness authority. The scheme helps only if proposal generation is cheap, acceptance is high, and target verification can exploit useful batching; it may provide no speedup when recurrent verification remains the dominant serial cost.

The speculative ratio is not the RL importance ratio. After exact verification, committed tokens are distributed according to p_i, so the rollout record stores $\log p_i(a_i)$ as the behavior log-probability. Proposal probabilities are metadata. If the learner later differentiates a different parallel or chunkwise

target, the RL update still needs Eq. (3.5). If the recurrent policy itself is the learning target, the learner needs the recurrent score in Eq. (3.13); proposal correction does not replace recurrent differentiation.

5 Recommended System Contracts

A production system should choose one of two execution-and-objective contracts rather than blur them.

1. **Recurrent-target mode.** The recurrent actor/deployment operator defines the target. Rollout stores recurrent post-transform probabilities, and the learner differentiates recurrent replay or a forward-and-backward equivalent implementation. A fast path may propose tokens, but exact recurrent verification determines what is committed. When actor and learner versions match, this removes the fixed-weight kernel gap and is literally on-policy. With actor-version lag, the remaining mismatch is ordinary off-policy staleness: token-local IS gives the local surrogate, whereas exact current-policy optimization requires the corresponding prefix or trajectory correction.
2. **Scalable parallel-target surrogate.** The recurrent actor generates data, while a parallel or chunkwise learner operator defines the target being differentiated. The learner uses the immutable actor probability and explicit recordwise importance ratios. This is mathematically exact for the local surrogate in Eq. (3.10); it is not automatically the full trajectory gradient or the gradient of recurrent deployment.

Both modes should use explicit support semantics, raw ratio diagnostics, FP32 recurrent-state baselines, an FP64 audit path, and separate controls for estimator variance and parameter movement.

REMEDY	EXACTNESS CLAIM	WHAT IT ADDRESSES	RESIDUAL LIMITATION
Recordwise importance ratio	Exact for the declared local surrogate under support	Kernel and actor-version action-law mismatch	Does not correct target history occupancy or behavior continuation
Prefix / trajectory products	Exact change of trajectory measure under support	Full target occupancy and continuation	Often prohibitive variance
Canonical execution rule	Exact at fixed weights if forward and backward contracts match	Removes implementation ambiguity	May reduce sequence parallelism
FP32/FP64 recurrent state	Numerical mitigation only	Reduces recurrent rounding drift	Cannot remove semantic schedule differences
Conditional proposal + recurrent verification	Exact recurrent sampling under the proposal contract	Accelerates rollout without changing target law	Needs actual proposal conditionals and sufficient acceptance

Table 2 No single technique solves distribution mismatch, gradient mismatch, support failure, variance, and numerical drift at once.

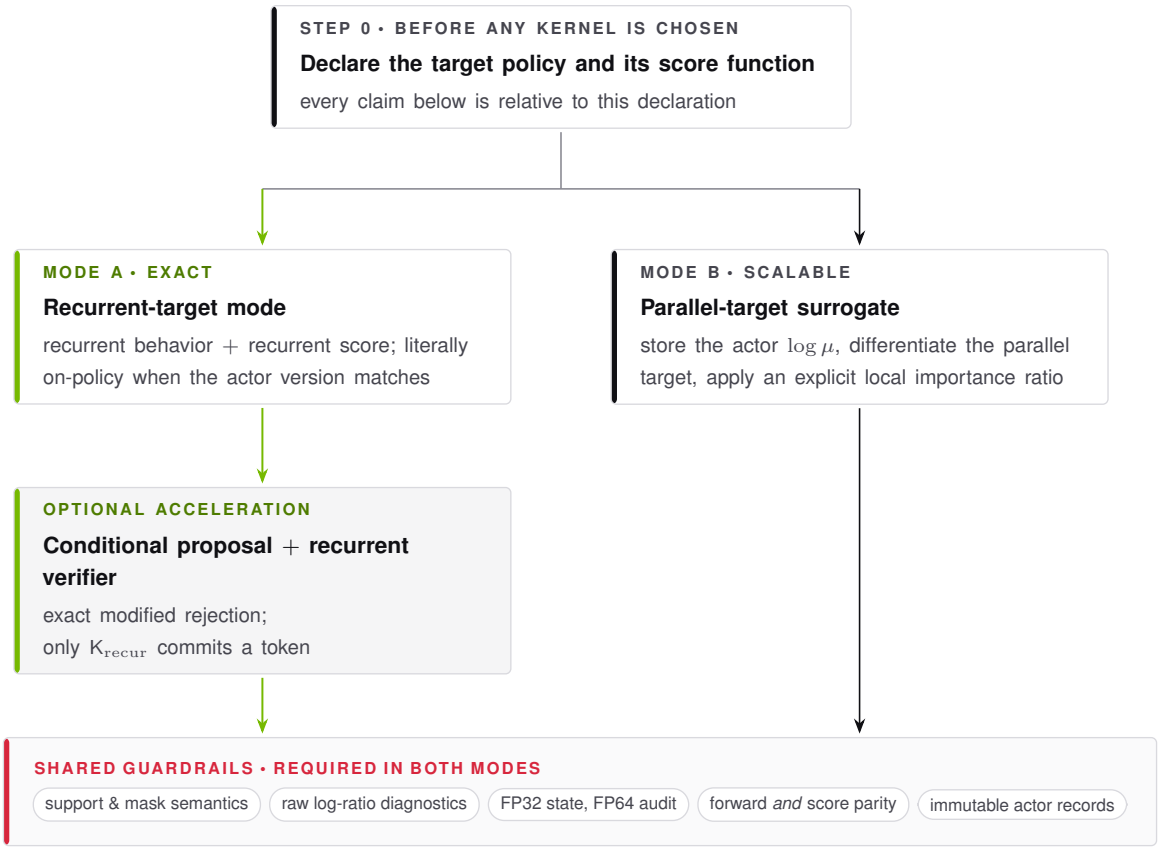

Figure 3 The target policy must be declared before kernels are chosen. Exact recurrent optimization and a scalable parallel-target surrogate are both coherent, but they make different claims; the guardrails apply to either.

6 Conclusion

Prefill–decode mismatch is not resolved by synchronizing model weights. A standard actor constructs state with prompt prefill and extends it with recurrent decode, while a learner often scores the response with a parallel teacher-forced pass. Linear attention, state-space models, and delta-rule memories add schedule-dependent state evolution, so chunkwise, scan, and recurrent implementations can induce different finite-precision policies.

The central correction is to separate three questions. Which post-transform distribution generated each token? Which executable policy is the learner differentiating? Is the claimed objective local, trajectory-level, or deployment-specific? Actor-emitted probabilities and explicit importance ratios answer the first question for off-policy data. Canonical recurrent or chunkwise replay answers the second when exact alignment is affordable. Prefix or trajectory products are needed for full change of measure. Higher precision reduces numerical drift but does not prove equality, and exact speculative verification accelerates sampling only under a valid proposal-probability contract.

The checkpoint name is therefore not the policy. The policy is the executable distribution, and an exact deployment-gradient claim additionally includes its score function.

References

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. arXiv preprint arXiv:2501.00663, 2025.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. arXiv preprint arXiv:2302.01318, 2023.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. arXiv preprint arXiv:2312.00752, 2023.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Bo Peng, Ruichong Zhang, Daniel Goldstein, Eric Alcaide, Xingjian Du, Haowen Hou, Jiaju Lin, Jiaxing Liu, Janna Lu, William Merrill, Guangyu Song, Kaifeng Tan, Saiteja Utpala, Nathan Wilce, Johan S. Wind, Tianyi Wu, Daniel Wuttke, and Christian Zhou-Zheng. RWKV-7 "Goose" with expressive dynamic state evolution. arXiv preprint arXiv:2503.14456, 2025.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *Proceedings of the 38th International Conference on Machine Learning*, pages 9355–9366, 2021.

State Spaces Team. Mamba reference implementation: precision notes. https://github.com/state-spaces/mamba, accessed August 9, 2026.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (learn at test time): RNNs with expressive hidden states. arXiv preprint arXiv:2407.04620, 2024.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *Advances in Neural Information Processing Systems 37*, 2024.

Ziyang Zhang, Xinheng Ding, Jiayi Yuan, Rixin Liu, Huizi Mao, Jiarong Xing, and Zirui Liu. Deterministic inference across tensor parallel sizes that eliminates training–inference mismatch. arXiv preprint arXiv:2511.17826, 2025.

Tianle Zhong, Neiwen Ling, Yifan Pi, Zijun Wei, Tianshu Yu, Geoffrey Fox, Peng Wu, and Xiao Yu. Diagnosing training inference mismatch in LLM reinforcement learning. arXiv preprint arXiv:2605.14220, 2026.

Appendix

A Auditing and Record Contracts

A.1 Minimal Fixed-Weight Parity Audit

A parity audit should be run independently of RL optimization. Freeze parameters, feed identical policy-visible prefixes to the actual rollout and learner paths, and separate four quantities.

1. **State parity:** compare prompt states, cache tensors, and recurrent boundary states before comparing logits.
2. **Forward-policy parity:** compare post-transform token distributions, including masks, temperature, truncation, and probability floors.
3. **Score parity:** for selected token log-probabilities, compare gradients, vector–Jacobian products, or finite-difference directional derivatives. Forward equality at one checkpoint does not establish this contract.
4. **Sampling parity:** verify that the probability recorded for the sampled token is exactly the probability used by the sampler after all transforms.

ARCHITECTURE	EXECUTABLE COMPARISONS
Softmax attention	Prompt prefill, recurrent KV-cached decode, differentiable recurrent replay, and full teacher-forced causal scoring.
Linear attention / SSM	Recurrent, scan, and chunkwise forms across chunk sizes, boundaries, state-materialization schedules, and precision.
DeltaNet	Tokenwise delta-rule updates versus the production chunkwise kernel, including FP32 and FP64 state references.
TTT / Titans-style memory	Online updates versus the exact mini-batch or segment rule, with identical read/write timing, momentum, decay, and resets.
Speculative rollout	Actual proposal conditionals, recurrent verifier distributions, acceptance draws, residual samples, committed states, and final actor probabilities.

Table 3 The audit compares executable traces and derivatives, not checkpoint names.

Vary precision, quantization, batch shape, tensor parallelism, sequence parallelism, and chunk size one factor at a time. When full distributions are available, report both directions of KL where finite, total variation, maximum and mean log-probability error, support-set disagreement, and sampled ratio quantiles. For backward checks, report relative gradient or VJP error on representative parameters and recurrent states.

A finite audit can falsify equality and quantify a tested configuration; it cannot prove equality over all histories. Any exact “on-policy” or “equivalent” claim should therefore state the execution contract and justify why equality extends beyond the tested prefixes. A tolerance-based audit supports a configuration-specific approximate claim and should name the tolerance and whether it covers forward distributions, score functions, or both.

A.2 Record and Logging Checklist

For every sampled policy token, preserve the actor-path post-transform log-probability and enough metadata to identify the behavior trace. At minimum, this includes actor version, rollout engine, attention or recurrence form, prefill/decode trace, chunk size or scan schedule, cache construction, precision, quantization, masks, sampling transform, and tensor/sequence-parallel layout.

The total log-ratio

$$\Delta^{\mathrm{total}}_{t,n,v} = \log \pi^{\mathsf{K}_{\mathrm{learn}}}_{\theta_n}(a_t \mid h_t) - \log \pi^{\mathsf{K}_{\mathrm{roll}}}_{\theta_v}(a_t \mid h_t) \tag{A.1}$$

is available from the learner score and immutable actor record. Its decomposition is

$$\Delta^{\mathrm{kernel}}_{t,n} = \log \pi^{\mathsf{K}_{\mathrm{learn}}}_{\theta_n}(a_t \mid h_t) - \log \pi^{\mathsf{K}_{\mathrm{roll}}}_{\theta_n}(a_t \mid h_t), \tag{A.2}$$

$$\Delta^{\mathrm{stale}}_{t,n,v} = \log \pi^{\mathsf{K}_{\mathrm{roll}}}_{\theta_n}(a_t \mid h_t) - \log \pi^{\mathsf{K}_{\mathrm{roll}}}_{\theta_v}(a_t \mid h_t). \tag{A.3}$$

Computing these two terms requires replaying the rollout operator at current weights and is usually best done on a sampled audit subset. Logging only Δ^{total} cannot identify whether refresh, kernel alignment, or precision is the appropriate intervention.

Exact per-history KL or TV requires either stored behavior distributions or a retained actor checkpoint that can reproduce them under the recorded operator. Sampled actor log-ratios can estimate behavior-averaged forward KL, but a single action is not an exact per-history divergence and does not determine TV or the reverse KL.

A.3 Minimal Off-Policy Record Contract

A rollout record is valid only if the learner can identify the distribution that produced every policy token. At minimum it contains:

1. the exact policy-visible prompt and history, sampled token IDs, termination markers, and policy-token mask;
2. the actor-emitted post-transform log-probability ℓ^{μ}_t for every policy token and the actor version v;
3. a per-token or segment-resolved execution signature: prefill/decode trace, recurrent or chunkwise form, chunk boundaries, state and parameter precision, quantization, cache construction, parallel layout, and reduction mode;
4. sampling metadata: temperature, admissible-action mask identity, probability floor, grammar or safety constraints, and any top-k/top-p/min-p truncation or stored support mask;
5. enough model provenance to reproduce audit distributions, such as a retained actor checkpoint or immutable content hash plus executable version;
6. for speculative rollout, the proposal law or actual conditionals q_i, verifier version, acceptance/rejection draws, residual sample, and final recurrent behavior probability $p_i(a_i)$; and
7. immutable reward, environment, verifier, termination, and sample-provenance metadata.

The learner may append diagnostics but must never replace ℓ^{μ}_t with a learner-side recomputation.

B Reference Implementation and Decision Checklist

B.1 Reference Pseudocode for Kernel-Corrected Learning

Algorithm 2 One kernel-corrected learner step.

Require: Immutable records with actor log-probabilities ℓ^μ and support/execution metadata; current learner θ_n; estimator mode; optional divergence rule.

Reject records whose history, support, or execution metadata violate the run contract.
Evaluate current target log-probabilities ℓ^p and score terms with the declared $\mathsf{K}_{\mathrm{learn}}$.
On valid policy tokens, compute raw $\log \rho \leftarrow \ell^p - \ell^\mu$ in FP64; reject nonfinite values and support violations.
if estimator mode is exact local IS **then**
    Set $\bar{\rho} \leftarrow \exp(\log \rho)$ without clipping.
else
    Apply the declared biased stabilization to obtain $\bar{\rho}$; retain raw $\log \rho$ for reporting.
end if
Compute a valid advantage A and the score loss in Eq. (3.17), replacing ρ by $\bar{\rho}$ in stabilized mode.
Save model, optimizer, and gradient-scaler state; take a tentative optimizer step.
if a divergence rule is declared and its required behavior distributions are available **then**
    Recompute the declared behavior-to-candidate divergence on audit histories.
    if the candidate violates the threshold **then**
        Restore all saved state; backtrack or reject the batch.
    else
        Commit the step.
    end if
else
    Commit the step and label sampled log-ratio statistics as diagnostics, not exact KL or TV.
end if
Log raw ratio moments, effective sample size, support failures, estimator mode, and audited kernel/version gaps separately.

B.2 Decision Checklist

QUESTION	REQUIRED ACTION
Is recurrent deployment the scientific target?	Differentiate recurrent replay or a verified forward-and-backward equivalent implementation; a recurrent ratio with a parallel score is not exact.
Do rollout and learner match only in forward probabilities?	The data may be on-policy for the learner distribution, but do not claim recurrent deployment-gradient equivalence without a score audit.
Is one chunk rule part of model semantics?	Freeze chunk size, boundaries, masks, read/write timing, momentum, decay, resets, precision, and backward implementation.
Must the learner retain a faster parallel target?	Store actor probabilities, enforce support, and use explicit IS; label token-local weighting as the local surrogate it is.
Does sampling use hard top-k/top-p/min-p?	Share and freeze an admissible support or label the run support-truncated; no ratio repairs actions with zero behavior probability.
Are recurrent states drifting?	Establish FP32 state/update arithmetic, compare against FP64, and separate rounding error from schedule error.
Can a fast path propose accurate blocks?	Use actual proposal conditionals with exact modified rejection and commit only recurrent target states.
Is the proposal non-autoregressive or correlated across positions?	Use a joint/block correction or specialized proof; token marginals are insufficient for standard speculative rejection.
Are ratios or divergences large?	Tighten actor lag, improve parity, reduce learner movement, or reject data; any ratio clipping must be declared as biased stabilization.

Table 4 A compact decision tree for choosing among the execution contracts.