FLASH ON-POLICY DISTILLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

On-policy distillation provides supervision at student-generated prefixes, before a response is complete. We present FLASH-OPD, a partial-rollout algorithm and overlapped execution architecture that separate the update horizon from the response horizon. Each round trains its *fresh suffixes* once, commits only after a successful update, and synchronizes before resuming unfinished trajectories. Historical tokens remain context rather than repeated targets. Ready suffixes trigger teacher evaluation while other requests decode; aligned target windows connect this concurrent service to a synchronized learner. A memory-bounded full-vocabulary loss supports this execution path. We specify the fresh-action alignment contract and the trade-off between hidden teacher work and added prefill, transfer, and synchronization. Across math, code, and agentic training, reported wall times improve by $2.02\times$, $1.85\times$, and $1.61\times$ over OPD, with primary task scores within 0.4 percentage points. Separate B200 measurements characterize teacher granularity, target delivery, and learner-backend costs without attributing the training gains to individual components.

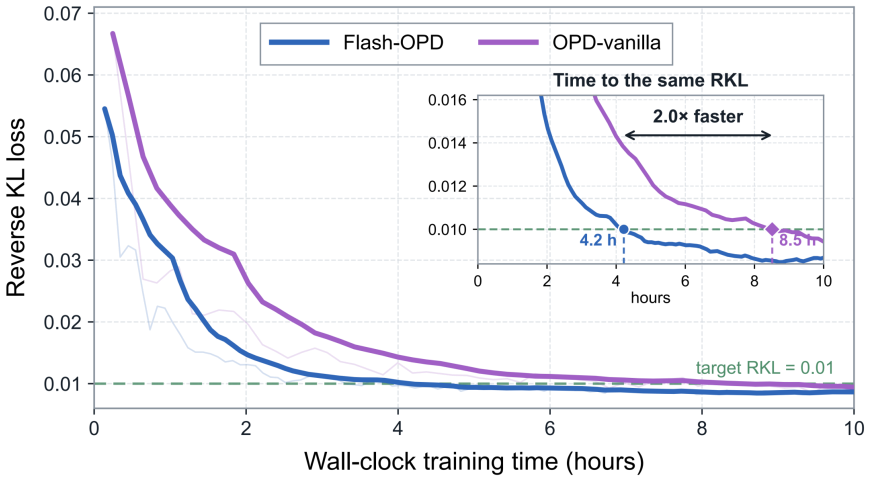


Figure 1: **Representative wall-clock convergence of Flash-OPD.** Flash-OPD reaches RKL 0.01 in 4.2 hours versus 8.5 hours for stage-serialized full-rollout OPD (*OPD-vanilla*), a $2.02\times$ time-to-target ratio. Faint/solid curves show raw values/10-step moving averages. This historical single-run trace predates the B200 candidates; it does not isolate components or establish task-quality equivalence.

1 INTRODUCTION

On-policy distillation (OPD) supervises a language model on its own generated prefixes (Agarwal et al., 2024; Gu et al., 2024; Ko et al., 2024). A teacher supplies a distribution at each state, without waiting for a terminal outcome. Full-vocabulary KL retains every next-token alternative, but requires the system to generate states, evaluate the teacher, deliver targets, and update the student.

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.

Supervision readiness need not coincide with response completion. For a causal teacher, a token's target depends on its prefix, not the eventual answer. Short-prefix training exploits this property (Lu & Thinking Machines Lab, 2025; Zhang et al., 2026a), while asynchronous OPD decouples completed-response collection from learning (Kang et al., 2026). We study a different execution boundary: train each newly generated suffix *before* continuing that same trajectory. This shortens the optimization horizon without treating the chunk boundary as EOS or permanently discarding the response tail.

A finer training unit requires a coordinated execution path. Earlier updates alone can add prefix prefill, teacher requests, and weight synchronization. Conversely, overlapping teacher calls with generation does not by itself make an unfinished suffix trainable: the learner still needs an explicit consumption and resumption boundary. We therefore design *partial updates and teacher overlap together*. The former determines when supervision can change the student; the latter exposes ready suffixes to the teacher without waiting for the rest of the round.

FLASH-OPD centers on two coupled designs:

- **Partial updates without discarding response tails.** A bounded fresh suffix is consumed in its generation round. One successful update, commit, and weight synchronization precede continuation; earlier tokens remain context, not repeated targets.
- **Readiness-driven teacher overlap.** Each ready suffix releases teacher work while peers decode. Aligned target windows and separate control/data paths connect this concurrency to the round's single update.

A memory-bounded full-vocabulary learner backend supports this architecture; it is not the central scheduling contribution. Full-distribution losses and hidden-state/tiled execution already appear in TRL, Liger, and Nitrobrew (Hugging Face, 2026b; Liger-Kernel Contributors, 2026; Pai et al., 2026). The contribution is the consume-before-continuation contract and the execution path that preserves it under teacher overlap, not new loss primitives. The evaluation puts training speed beside task quality, then examines teacher granularity, delivery costs, and loss-backend trade-offs. The key systems question is whether overlap saves more time than partial updates add in prefill, requests, and synchronization.

2 PROBLEM SETUP

Sampling and supervision. For a prompt $x \sim \mathcal{D}$ and nonempty response $y = (y_1, \dots, y_T)$, $p_\theta(y \mid x) = \prod_{t=1}^{T} p_\theta(y_t \mid s_t)$, where $s_t = (x, y_{<t})$ is the prefix state and y_t is its next-token action. We distinguish this model distribution from its rollout law $\pi_\theta = \mathsf{Decode}_\psi(p_\theta)$, where fixed settings ψ include decoding temperature τ_{dec} and any support filtering. The distillation temperature τ below is a separate parameter.

On-policy distillation. OPD samples from a detached student snapshot $\bar{\theta}_k = \mathrm{sg}(\theta_k)$ and evaluates a fixed teacher at the visited states. Let $q_M^{(\tau)}(\cdot \mid s) = \mathrm{softmax}(z_M(s)/\tau)$ for $M \in \{\theta, \mathrm{T}\}$ and $\tau > 0$. We use the token-local reverse KL

$$\ell_{\mathrm{RKL}}(\theta; s) = D_{\mathrm{KL}}\Big(q_\theta^{(\tau)}(\cdot \mid s) \,\Big\|\, q_{\mathrm{T}}^{(\tau)}(\cdot \mid s)\Big) = \sum_{v \in \mathcal{V}} q_\theta^{(\tau)}(v \mid s) \log \frac{q_\theta^{(\tau)}(v \mid s)}{q_{\mathrm{T}}^{(\tau)}(v \mid s)}. \tag{1}$$

Models share a vocabulary and aligned token positions; no additional τ^2 multiplier is included. Full vocabulary refers to the loss support, not necessarily the decoding support. Gradients hold sampled token sequences and teacher targets fixed, without differentiating through generation. Token-mean and response-mean aggregation need not agree; our fresh-token normalization is specified in Section 3.1.

Hidden states as a transport representation. For linear heads, $z_M(s) = W_M h_M(s)$ with $W_M \in \mathbb{R}^{|\mathcal{V}| \times d_M}$. Transferring the teacher state $h_{\mathrm{T}}(s)$ and keeping its frozen head at the learner permits local logit reconstruction. This remains *output-space* KL; hidden widths may differ, and hidden vectors are not directly matched.

Three execution boundaries. A stage-serialized implementation exposes three costs: *completion* delays the update until the last response; *service dispatch* exposes teacher work after rollout; *distribution materialization* creates $N \times |\mathcal{V}|$ intermediates for N states. These are structural costs, not a measured bottleneck ranking. The following design addresses them together.

3 Flash On-Policy Distillation

Flash-OPD coordinates a persistent trajectory pool, concurrent rollout/teacher services, and a synchronized learner. Its defining unit is the fresh suffix, not a completed response. Within round k, requests may finish and dispatch teacher work independently, but all use the fixed snapshot $\bar{\theta}_k = \mathrm{sg}(\theta_k)$. One successful update, commit, and weight synchronization then release unfinished trajectories for continuation.

3.1 Partial Updates with Persistent Trajectories

Maintain a pool of at most B unfinished trajectories, refilling released slots between rounds. A scheduled trajectory $i \in \mathcal{A}_k$ has $b_{i,k}$ committed response tokens and carries context $c_{i,k} = (x_i, y_{i,1:b_{i,k}})$, with $b_{i,k} = 0$ on admission. For quantum H and cumulative cap $L_{\max}$, it generates a suffix $u_{i,k}$ under $\pi_{\bar{\theta}_k}$ of length $m_{i,k} \le \min(H, L_{\max} - b_{i,k})$. Its fresh positions are $\mathcal{I}_{i,k} = \{b_{i,k}+1, \ldots, b_{i,k}+m_{i,k}\}$; $M_{i,k}$ is their indicator. Chunk boundaries are not EOS.

A separate optimization horizon. H limits new tokens before an update, not wall time or total response length. For $H \ge L_{\max}$, a new response completes before its first update; smaller quanta permit intermediate updates without discarding its tail. Updates change future continuations, not committed tokens. The resulting response can mix snapshots, and equal token budgets need not imply equal update counts or visited states.

For the set $\mathcal{N}_k$ of fresh (i,t) positions in the round and $Z_k = \sum_i m_{i,k} > 0$, the learner minimizes

$$\widehat{\mathcal{L}}^{(k)}_{\mathrm{Flash}}(\theta; \bar{\theta}_k) = \frac{1}{Z_k} \sum_{(i,t)\in\mathcal{N}_k} \ell_{\mathrm{RKL}}(\theta; s_{i,t}). \tag{2}$$

Historical positions remain causal context with $M_{i,k}(t) = 0$: the mask removes their direct loss terms, not gradient paths through prefix computation. Microbatches share the global denominator Z_k. Gradient accumulation is allowed, but only one optimizer step and no replay occur before the suffixes are committed (Appendix B.1).

Consume, commit, then continue. After a successful update, advance each `trained_upto` offset by $m_{i,k}$, retire completed responses, and synchronize weights before resuming unfinished contexts. This ordering gives two execution invariants: every loss-bearing action was generated by the update's snapshot, and committed suffix intervals cover each trained token once. A failed or skipped update must not advance offsets. This is *fresh-action alignment*: actions use the current snapshot but condition on historical prefixes. It does not imply current-policy prompt-rooted state occupancy or an unbiased full-response gradient. Masking prior partial-rollout tokens during resumption (slime Contributors, 2026) does not establish that they were already trained; here each committed suffix was consumed in its generation round (Figure 2).

Failure and finite-budget behavior. An all-empty batch takes no optimizer step, and a failed or skipped update must not commit or retire pending suffixes. Empty replies retire only with explicit EOS/cap status, never on a service error. Nondivisible tail groups are excluded before generation. A finite budget may leave open trajectories; this ordering contract does not provide crash recovery.

3.2 Within-Round Overlap Architecture

Partial updates determine *when to learn*; per-suffix dispatch determines *when teacher work becomes runnable*. The two boundaries are distinct: teacher evaluation can start before batch collection finishes, but continuation cannot cross the update/weight-sync barrier (Figure 3).

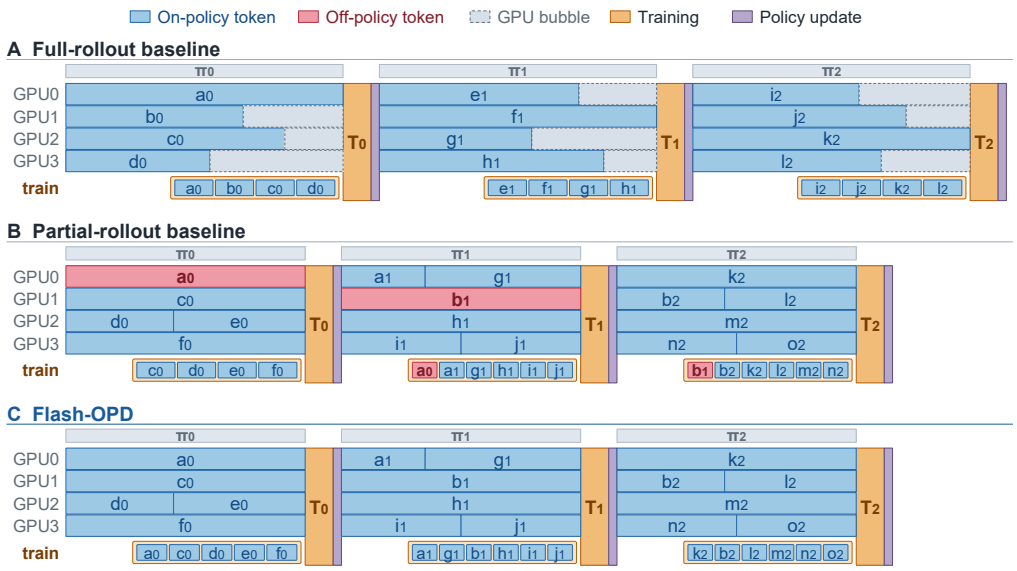


Figure 2: **Policy alignment under full and partial rollouts.** **(A)** Full rollout waits for completion. **(B)** Unmasked partial rollout trains earlier-policy tokens (red). **(C)** Flash-OPD consumes current suffixes (blue), retaining history as context. Subscripts mark snapshots; orange/purple mark training/updates. This is a schematic, not a measured timeline or a depiction of every partial baseline.

Per-suffix dispatch. Each sample task in `verl`'s agent loop invokes the teacher as soon as its suffix is ready, before the batch gather (Helwig, 2026). Teacher work can overlap other requests' decoding on disjoint GPUs.

One update boundary. All current-round tasks finish before learning. The learner then retrieves the target windows and updates once; no cross-version replay queue is used.

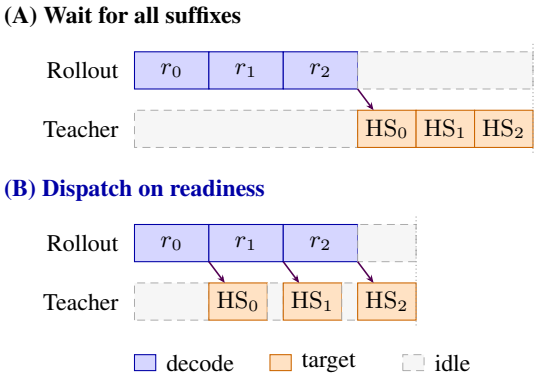


Figure 3: **Within-round teacher overlap.** A logical schedule, not a measured timeline. Teacher batching and replicas are omitted; deferred payload transfer remains a separate cost.

Overlap is within a round, not across policy versions. Independent teacher dispatch retains one learner barrier. Early jobs can overlap decoding, but late suffixes or a backed-up service extend the round. Batching and replicas change this exposed tail, not the update contract; avoiding cross-version replay does not make teacher service free.

A boundary-aligned target window. The teacher receives the full visible sequence $c_{i,k} \| u_{i,k}$, but returns only the $m_{i,k}$ predictive states needed by the loss. For an extended prompt of $P_{i,k}$ tokens, their zero-based positions are

$$\mathcal{J}_{i,k} = [P_{i,k} - 1,\ P_{i,k} + m_{i,k} - 1). \tag{3}$$

The last prompt state predicts the first suffix token; the final suffix state is not needed. The pooling teacher returns final states without additional embedding normalization; its linear head remains frozen

at the learner. Cache hits that omit the boundary state trigger a retry without cache reads. Teacher prefixes can be reused while its weights remain fixed; student KV states must be invalidated after an update.

For m target positions, transferring d_{T}-wide states rather than V-wide logits changes the payload from $b_z mV$ to $b_h m d_{\mathrm{T}}$ bytes, where b_z, b_h are their element sizes. The ratio $b_h d_{\mathrm{T}}/(b_z V)$ excludes receipts and broadcasts; the learner must still hold the teacher head.

Separate control and data paths. Teacher outputs are staged in pinned memory; the batch transports a receipt identifying the model, endpoint, tensor, and window. The owning learner redeems the window over NCCL, once per shared tensor/context-parallel row group, then broadcasts it. The driver handles receipts rather than tensor payloads. Deferred transfer and learner projections are not assumed to overlap rollout. Appendix A details the protocol and cache recovery.

3.3 Execution Schedule and Critical Path

Algorithm 1 makes the generation–service–update ordering explicit. It exploits within-round concurrency without letting the learner update beneath outstanding suffixes. Empty rows, dispatch divisibility, and finite-budget handling follow Appendix A.

Algorithm 1 Flash-OPD: overlap ready supervision, update, then continue

Require: Student p_θ; fixed teacher and output head W_{T}; prompt stream $\mathcal{D}$; pool capacity B; quantum H; cap $L_{\max}$

Initialize an empty trajectory pool; load W_{T} at the learner
for round $k = 0, 1, \ldots$ within the training budget **do**
  Refill pool slots; select dispatchable $\mathcal{A}_k$; **break** if empty
  $\bar{\theta}_k \leftarrow \mathrm{sg}(\theta_k)$ ▷ one fixed behavior snapshot
  for all $i \in \mathcal{A}_k$ as concurrent sample tasks **do**
    $h_{i,k} \leftarrow \min(H, L_{\max} - b_{i,k})$
    Generate $u_{i,k}$ from $c_{i,k}$ under $\pi_{\bar{\theta}_k}$, up to $h_{i,k}$ tokens
    Record length $m_{i,k}$ and the fresh-position mask $M_{i,k}$
    if $m_{i,k} > 0$ **then**
      $\rho_{i,k} \leftarrow$ await staged-target receipt for $c_{i,k} \| u_{i,k}$ ▷ other tasks keep generating
    end if
  end for
  Gather sample tasks and the receipts $\rho_{i,k}$ for nonempty suffixes
  if $Z_k = \sum_i m_{i,k} > 0$ **then**
    Redeem teacher windows at their owning learner ranks
    Compute student states and full-vocabulary KL using W_θ, W_{T}
    Accumulate gradients on $M_{i,k} = 1$ positions with global denominator Z_k
    Take one optimizer step; **abort** on failure or skip before pool mutation
    Commit trained suffixes and retire completed responses
    Synchronize updated student weights before the next rollout round
  end if
  Retain unfinished contexts; retire any zero-token terminal entries
end for

When does finer granularity save time? For a matched workload and allocation, let S_{F} and S_{S} sum all stage costs for Flash and the serialized baseline, including dispatch, prefill, staging, learning, and weight synchronization. Let O_{net} be Flash's reduction from its stage sum through overlap. Then

$$C_{\mathrm{Flash}} - C_{\mathrm{serial}} = (S_{\mathrm{F}} - S_{\mathrm{S}}) - O_{\mathrm{net}} < 0 \quad \Longleftrightarrow \quad O_{\mathrm{net}} > S_{\mathrm{F}} - S_{\mathrm{S}}. \tag{4}$$

Smaller H exposes supervision earlier but adds requests, repeated student prefill, and weight synchronization. A fixed teacher may reuse prefix state; the updated student cannot. Deferred transfer and loss evaluation remain exposed learner work. Thus H is an algorithm–system trade-off, not a knob to minimize unconditionally. Section 4 examines these costs; the accounting condition alone does not predict time-to-quality.

Repeated prefix work. Without cache reuse, a response split into n equal H-token suffixes after a P-token prompt requests $nP + Hn(n+1)/2$ teacher positions, versus $P + nH$ for full-response evaluation. This is a position count, not an attention FLOP model. Cache misses, eviction, and boundary retries retain part of that cost; student prefix recomputation after each update is unavoidable in this design.

Two independent scheduling controls. Full responses can use teacher overlap; partial collection can defer teacher work to a batch barrier. Isolating these choices requires fixed placement, target representation, and loss backend. Changing the backend simultaneously would conflate scheduling with loss-computation speed.

Supporting full-vocabulary learning. The learner must consume full-distribution targets without making each suffix batch require dense $N \times V$ logits. We use a vocabulary-tiled backend with online statistics and backward recomputation (Milakov & Gimelshein, 2018; Pai et al., 2026). For N evaluated states, token chunk C, and vocabulary tile B_v, its working set is

$$\text{LogitWorkspace} = \mathcal{O}(CB_v), \qquad \text{SavedStatistics} = \mathcal{O}(N), \tag{5}$$

excluding inputs, resident heads, the dense student head gradient, decoder activations, and optimizer state. The parameters C, B_v do not set the rollout quantum H. This backend preserves the supplied full-vocabulary loss in real arithmetic; finite-precision execution and native-teacher alignment require separate checks. Its FSDP/vocabulary-parallel integration, derivation, and opt-in precision policies are detailed in Appendices B.2 and D. Section 4.3 evaluates its resource trade-off, not an independent claim about partial-rollout speed.

4 Evaluation

The main comparison pairs task quality with training speed: reducing wall time is useful only if the resulting model retains OPD's task performance. We then examine the service and delivery costs introduced by partial updates, followed by the full-vocabulary learner backend that supports the architecture.

4.1 Task Quality and Training Speed

Table 1 reports end-to-end speedups of $2.02\times$, $1.85\times$, and $1.61\times$ for math, code, and agentic training. OPD and FLASH-OPD both round to a Macro score of 67.0%; primary-score differences range from -0.4 to $+0.4$ points. These observed scores accompany reduced training time, but do not establish statistical equivalence or isolate component contributions. The B200 study below measures component costs separately.

Representative optimization progress. Figure 1 reaches RKL 0.01 in 4.2 rather than 8.5 hours against stage-serialized full-rollout OPD, using a 10-step moving average. The $2.02\times$ ratio concerns that metric, not task-quality equivalence or isolated overlap gains. This historical trace predates the B200 study and lacks complete run provenance. The latter's online screen completed no optimizer update; its component measurements therefore diagnose execution costs rather than establish a matched end-to-end speedup.

4.2 Teacher Service at Partial Boundaries

Shorter quanta create earlier opportunities to dispatch supervision, but also more teacher requests. To isolate this cost, we replay four fixed 1,024-token responses at $H \in \{128, 256, 512, 1024\}$, with three repeats and cache off/on for each of two teachers. No student decoding or learning is timed in this probe; it measures the service work that overlap must accommodate.

The primary hybrid teacher requires chunked prefill in the imported serving version, unlike the production default. Its probe uses TP1, eager execution, one active request, and a resolved 1,056-token cache-alignment block. The short horizons therefore test limited reuse, not batched production teacher capacity.

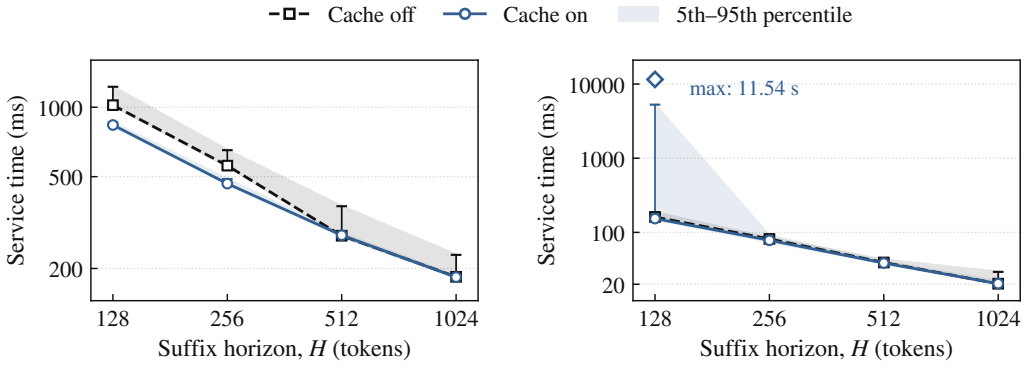
Table 1: **Task quality and end-to-end training speed. (a)** Wall time and speedup $T_{\mathrm{OPD}}/T_{\mathrm{method}}$ at matched supervised-token budgets and total accelerator allocations; larger speedup is better. **(b)** Task scores (%); Macro averages the five primary scores, excluding pass columns; Δ is Flash-OPD minus OPD in points, computed from displayed scores. Olym. and SWE-Bench are evaluated once, without separate pass metrics. Bold: best displayed student scores, including ties.

(a) End-to-end training efficiency

Method	Math		Code		Agentic	
	Time (h)↓	**Speedup↑**	Time (h)↓	**Speedup↑**	Time (h)↓	**Speedup↑**
OPD	8.5	1.00×	6.3	1.00×	12.7	1.00×
FLASH-OPD	**4.2**	**2.02×**	**3.4**	**1.85×**	**7.9**	**1.61×**

(b) Downstream task quality (%)

Method	Math					Code		Agentic	Summary
	AIME 24		AMC		Olym.	LCB		SWE-Bench	Macro
	Avg @32	pass @32	Avg @4	pass @4	Avg @1	Avg @4	pass @4	Resolved	Score
Teacher	77.4	93.3	93.7	97.6	66.2	54.6	62.7	55.0	69.4
Initial student	73.3	90.0	87.0	94.0	62.1	49.5	**60.6**	40.4	62.5
OPD	**74.3**	90.0	**92.5**	95.2	65.3	53.6	60.2	**49.2**	**67.0**
FLASH-OPD	**74.3**	**93.3**	92.1	**96.0**	**65.7**	**53.9**	60.2	49.0	**67.0**
Δ vs. OPD	0.0	+3.3	−0.4	+0.8	+0.4	+0.3	0.0	−0.2	0.0

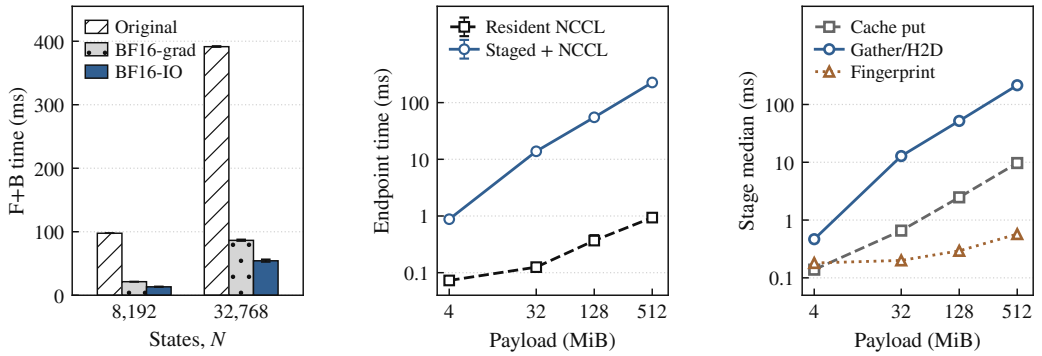


(a) Qwen3.6-35B-A3B (b) Qwen3-4B

Figure 4: **Shorter update horizons add teacher work.** Serial service per 1,024-token response: medians and 5th–95th trajectory percentiles (four samples, three repeats), not confidence intervals. Both axes are logarithmic. The hybrid teacher uses a compatibility probe; the diamond marks Qwen3-4B's 11.54 s maximum. These are not decoding-overlap measurements.

Figure 4 shows that shorter horizons increase aggregate serial teacher-service time. Caching reduces returned rows more than latency; for Qwen3-4B at $H = 128$, the cached median is 153.54 ms but one trajectory takes 11.54 s. Cached/no-cache states differ by up to 0.0300 relative L_2, with unmeasured loss impact. Because the update waits for every required target, a long service tail can remain exposed even when other teacher work overlaps decoding. Caching cannot simply be assumed to make repeated prefix service free.

Target delivery remains on the update path. The learner retrieves target payloads after collection; per-suffix teacher dispatch does not automatically hide this transfer. Figure 5 separates delivery cost from distributed head-loss execution.



(a) TP4 loss (b) Transfer paths (c) Staging costs

Figure 5: **Distributed loss and target-delivery costs.** (a) TP4 head loss including NCCL reductions. (b) Resident versus staged transfer. (c) Separately measured staging medians; these are not additive parts of the endpoint median. Panels (b,c) use log axes. Whiskers in (a,b) show 5th–95th iteration percentiles, not independent-run uncertainty. Ray control work is excluded.

For a 128 MiB payload, resident NCCL takes 0.368 ms, versus 55.056 ms for the staged path; gather/H2D accounts for 51.946 ms of median substage time. Smaller payloads do not remove host staging. Separately, with four vocabulary shards the BF16-IO backend takes 54.27 ms and 3.86 GiB peak per rank at 32K states: 7.21× faster than original Flash on the *same four devices*, not a full-training scaling result.

4.3 Supporting Full-Vocabulary Learner Backend

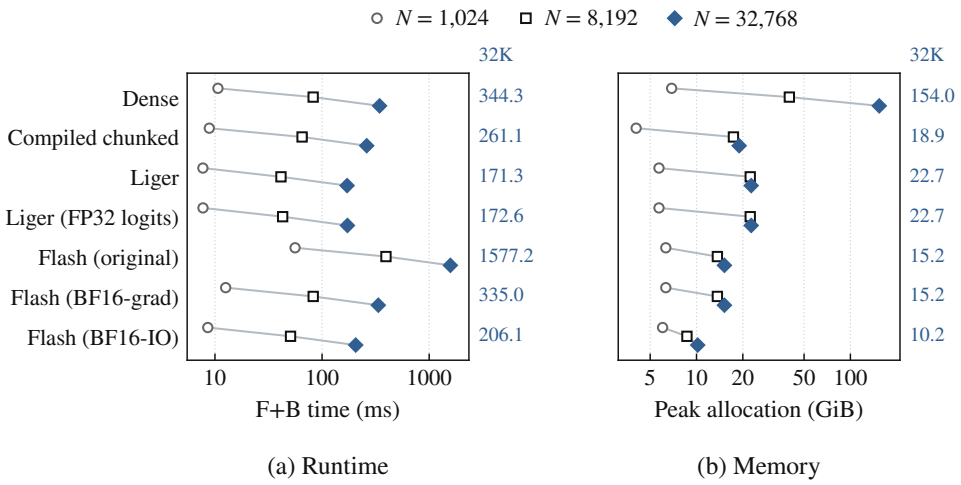
This benchmark characterizes the loss backend, independently of the partial schedule. Qwen3.5-4B and Qwen3.6-35B-A3B supply student/teacher states of widths $2{,}560/2{,}048$, with $V = 248{,}320$. The bank contains 32,768 teacher-forced corpus states; smaller N uses nested subsets, not on-policy samples. Token-mean full-vocabulary RKL at $\tau = 1$ requires both student hidden and head gradients. One-B200 timings use TF32 off, 20 warmups, and 300 iterations in one process; memory is measured separately. References use latency-selected configurations, and compiled chunking is not a particular TRL release (Appendix D).

At 32K states, BF16-IO takes 206.08 ms and 10.18 GiB. Compared with Liger's 171.26 ms and 22.67 GiB, this exchanges 20.3% more latency for 55.1% less memory. Against dense and compiled chunking it is 1.67× and 1.27× faster, respectively. Original Flash is slower than the references; the opt-in policies improve that backend rather than establishing a scheduling gain. Across six candidate/size checks, loss relative error is below 3.2×10^{-6} and gradient relative L_2 error below 0.0032. Exact values, profiling, and precision restrictions remain in Appendix D.

5 Related Work

On-policy distillation. GKD, MiniLLM, and DistiLLM study student-generated states, divergences, and trajectory reuse (Agarwal et al., 2024; Gu et al., 2024; Ko et al., 2024). Recent analyses distinguish teacher compatibility, token-level alignment, and task outcomes (Jin et al., 2026; Li et al., 2026). Flash-OPD changes execution granularity rather than proposing a new divergence; full support does not by itself establish better task quality.

Shortened and asynchronous rollouts. Prefix OPD and horizon-scheduled variants truncate the generated training response (Zhang et al., 2026a;b); NPD reuses packed, teacher-annotated responses



(a) Runtime (b) Memory

Figure 6: **Supporting loss-backend memory–latency trade-off.** One B200, TF32 off, both student hidden/head gradients. Marker shapes identify workload sizes, connected within each backend; numeric columns report 32K. Horizontal axes are logarithmic. Latency whiskers are 5th–95th iteration percentiles, not confidence intervals. Peak excludes decoder/optimizer; BF16 variants are opt-in.

between refreshes (Rang et al., 2026). AsyncOPD already pauses and resumes in-flight prefixes but learns from completed responses through a bounded backlog (Kang et al., 2026). APRIL similarly retains unfinished responses across policy updates; slime can mask an existing response prefix before resumed generation (Zhou et al., 2025; slime Contributors, 2026). Neither resumption nor setting that mask implies an immediate update on the interrupted suffix. AsyncOPD's outstanding-work bound also differs from a hard per-sample policy-age cap. Our distinction is consuming each suffix before continuation, not continuation alone. The design complements serving, placement, and stage-overlap systems such as vLLM, HybridFlow, and RLHFuse (Kwon et al., 2023; Sheng et al., 2025; Zhong et al., 2025).

Full-distribution execution. TRL and Liger already support memory-efficient full-vocabulary distillation (Hugging Face, 2026b; Liger-Kernel Contributors, 2026). TRL's synchronous path checkpoints full-vocabulary token chunks; its separate asynchronous path uses a completion queue and reduced teacher information (Hugging Face, 2026a). Concurrency and target support are therefore independent comparison axes, not interchangeable labels for the same baseline. Nitrobrew combines hidden-state transport, local teacher-head reconstruction, and online vocabulary-tiled divergence (Pai et al., 2026); these are not standalone novelty claims here. We integrate them with suffix-aligned targets and an explicit update/commit boundary, building on streaming normalization and recomputation (Milakov & Gimelshein, 2018; Dao et al., 2022; Wijmans et al., 2025). Appendix C details the distinctions; no benchmark ranking against Nitrobrew is implied.

6 Conclusion

FLASH-OPD separates response completion from the decision to learn. Its partial-update algorithm consumes fresh suffixes before continuation, while its overlap architecture dispatches ready teacher work within a synchronized round. Full-vocabulary target delivery and a memory-bounded learner support this execution path. The resulting trade-off is between earlier supervision and added prefill, service, and synchronization; component measurements expose these costs but do not replace matched quality–speed controls. The central design principle is to coordinate when supervision becomes available, when it is consumed, and when generation resumes—not merely accelerate the loss in isolation.

AI USE STATEMENT

Generative AI assisted with manuscript polishing, literature review, mathematical formulation and proof exposition, experimental design, data preparation, code implementation and experiment execution, result interpretation, and figure preparation. The authors remain responsible for verifying the AI-assisted content, implementation, and reported results.

REPRODUCIBILITY STATEMENT

Section 3 specifies the execution contract and algorithm, and Appendix A documents the independent rollout and kernel controls, target-window alignment, and resource accounting. Appendix B gives distributed normalization and the kernel derivation; Appendix B.3 specifies numerical checks and measurement requirements. Section 4 distinguishes the representative convergence trace from component benchmarks and states the limits of the available evidence. Software defaults are not presented as the resolved configuration or provenance of that trace. Appendix D documents the B200 component protocols, selected configurations, numerical comparisons, and measurement scope. Preserved raw records distinguish timing iterations from independent runs; the study makes no downstream-quality claim from component timings.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, 2024. URL https://iclr.cc/virtual/2024/poster/19484.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, volume 35, pp. 16344–16359, 2022. doi: 10.52202/068431-1189. URL https://papers.nips.cc/paper_files/paper/2022/hash/67d57c32e20fd0a7a302cb81d36e40d5-Abstract-Conference.html.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. MiniLLM: Knowledge distillation of large language models. In *International Conference on Learning Representations*, 2024. URL https://iclr.cc/virtual/2024/poster/19420.

Jacob Helwig. On-policy distillation (OPD). verl documentation, 2026. URL https://verl.readthedocs.io/en/latest/algo/opd.html. Updated May 26, 2026; accessed September 26, 2026.

Hugging Face. TRL: Generalized knowledge distillation trainer. Software documentation, version 0.19.1, 2025. URL https://huggingface.co/docs/trl/v0.19.1/en/gkd_trainer. Version released July 8, 2025; accessed September 26, 2026.

Hugging Face. TRL: Asynchronous distillation trainer. Software source, 2026a. URL https://github.com/huggingface/trl/blob/8fbe6252a3d5aafe80e0c27477082cf5df131e3d/trl/experimental/async_distillation/async_distillation_trainer.py. Revision 8fbe625; accessed September 26, 2026.

Hugging Face. TRL: Distillation trainer. Software source, 2026b. URL https://github.com/huggingface/trl/blob/8fbe6252a3d5aafe80e0c27477082cf5df131e3d/trl/trainer/distillation_trainer.py. Revision 8fbe625; accessed September 26, 2026.

Woogyeol Jin, Taywon Min, Yongjin Yang, Dennis Wei, Yi Zhou, Swanand Ravindra Kadhe, Nathalie Baracaldo, and Kimin Lee. Entropy-aware on-policy distillation of language models. In *International Conference on Machine Learning*, 2026. URL https://icml.cc/virtual/2026/poster/64855.

Wonjun Kang, Kevin Galim, Seunghyuk Oh, Minjun Kang, Sanghyun Park, Donghoon Kim, Minjae Lee, Minseo Kim, Rishabh Tiwari, Yuchen Zeng, Hyung Il Koo, and Kangwook Lee. AsyncOPD: How stale can on-policy distillation be? *arXiv preprint arXiv:2606.24143*, 2026. URL https://arxiv.org/abs/2606.24143v1.

Jongwoo Ko, Sungnyun Kim, Tianyi Chen, and Se-Young Yun. DistiLLM: Towards streamlined distillation for large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 24872–24895. PMLR, 2024. URL https://proceedings.mlr.press/v235/ko24c.html.

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 Symposium on Operating Systems Principles*, pp. 611–626, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Yaxuan Li, Yuxin Zuo, Bingxiang He, Jinqian Zhang, Chaojun Xiao, Cheng Qian, Tianyu Yu, Huan-ang Gao, Wenkai Yang, Zhiyuan Liu, and Ning Ding. Rethinking on-policy distillation of large language models: Phenomenology, mechanism, and recipe. *arXiv preprint arXiv:2604.13016*, 2026. URL https://arxiv.org/abs/2604.13016v2.

Liger-Kernel Contributors. Liger-Kernel: Fused linear JSD implementation. Software source, `fused_linear_jsd.py`, 2026. URL https://github.com/linkedin/Liger-Kernel/blob/ad2436e80e1a93729aa34de5be7f3ccfa847da52/src/liger_kernel/ops/fused_linear_jsd.py. Revision ad2436e; accessed September 26, 2026.

Kevin Lu and Thinking Machines Lab. On-policy distillation. Thinking Machines Lab: Connectionism, 2025. URL https://thinkingmachines.ai/blog/on-policy-distillation/.

Maxim Milakov and Natalia Gimelshein. Online normalizer calculation for softmax. *arXiv preprint arXiv:1805.02867*, 2018. URL https://arxiv.org/abs/1805.02867.

Dhruv Pai, Timor Averbuch, Alec Dewulf, Ben Keigwin, and Ashley Zhang. Nitrobrew: Fast, lossless distillation for free. Tilde Research technical blog, 2026. URL https://blog.tilderesearch.com/blog/nitrobrew. Published April 28, 2026; accessed September 26, 2026.

Miao Rang, Zhenni Bi, Hang Zhou, Kai Han, Xuechun Wang, An Xiao, Xinghao Chen, Yunhe Wang, and Hanting Chen. Near-policy: Accelerating on-policy distillation via asynchronous generation and selective packing. *arXiv preprint arXiv:2605.05940*, 2026. URL https://arxiv.org/abs/2605.05940v1.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 1279–1297, 2025. doi: 10.1145/3689031.3696075. URL https://doi.org/10.1145/3689031.3696075.

slime Contributors. slime: Partial-rollout implementation. Software source, `sglang_rollout.py`, 2026. URL https://github.com/THUDM/slime/blob/4c193f1f37509cca70f0e88807a9305b70f63f4e/slime/rollout/sglang_rollout.py. Revision 4c193f1; accessed September 26, 2026.

Philippe Tillet, H. T. Kung, and David Cox. Triton: An intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 10–19. ACM, 2019. doi: 10.1145/3315508.3329973. URL https://doi.org/10.1145/3315508.3329973.

Erik Wijmans, Brody Huval, Alexander Hertzberg, Vladlen Koltun, and Philipp Krähenbühl. Cut your losses in large-vocabulary language models. In *International Conference on Learning Representations*, 2025. URL https://iclr.cc/virtual/2025/poster/30424.

Dongxu Zhang, Zhichao Yang, Sepehr Janghorbani, Jun Han, Andrew Ressler, II, Qian Qian, Gregory D. Lyng, Sanjit Singh Batra, and Robert E. Tillman. Fast and effective on-policy distillation from reasoning prefixes. *arXiv preprint arXiv:2602.15260*, 2026a. URL https://arxiv.org/abs/2602.15260v1.

Yaocheng Zhang, Jiajun Chai, Yuqian Fu, Songjun Tu, Xiaohan Wang, Wei Lin, Guojun Yin, Qichao Zhang, Yuanheng Zhu, and Dongbin Zhao. Are full rollouts necessary for on-policy distillation? *arXiv preprint arXiv:2605.31490*, 2026b. URL https://arxiv.org/abs/2605.31490v2.

Yinmin Zhong, Zili Zhang, Bingyang Wu, Shengyu Liu, Yukun Chen, Changyi Wan, Hanpeng Hu, Lei Xia, Ranchen Ming, Yibo Zhu, and Xin Jin. Optimizing RLHF training for large language models with stage fusion. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pp. 489–503. USENIX Association, 2025. URL https://www.usenix.org/conference/nsdi25/presentation/zhong.

Yuzhen Zhou, Jiajun Li, Yusheng Su, Gowtham Ramesh, Zilin Zhu, Xiang Long, Chenyang Zhao, Jin Pan, Xiaodong Yu, Ze Wang, Kangrui Du, Jialian Wu, Ximeng Sun, Jiang Liu, Qiaolin Yu, Hao Chen, Zicheng Liu, and Emad Barsoum. APRIL: Active partial rollouts in reinforcement learning to tame long-tail generation. *arXiv preprint arXiv:2509.18521*, 2025. URL https://arxiv.org/abs/2509.18521v3.

A Implementation Details

A.1 Runtime Contract and Configuration

Flash-OPD adds a trainer, trajectory pool, teacher-window service, and FSDP/Megatron adapters to `verl` through feature-local compatibility hooks. Direct distillation disables task rewards and policy-gradient estimation. Table 2 records independent software controls, not the resolved configuration of the historical convergence trace.

Table 2: **Independent execution controls.** Default and strict-mode settings.

Control	Setting	Operational role
Rollout and update contract		
Rollout quantum H	512	New tokens per trajectory and round
Pool capacity B	Training batch size (if unset)	Concurrent resident trajectories
Response cap $L_{\max}$	Configured limit	Cumulative across all suffixes
Optimizer minibatches	1	One update per collected batch
Replay epochs	1	No repeated suffix optimization
Loss working set		
Kernel token chunk C	8,192	Projection/reduction working set
Vocabulary tile B_v	32,768	Logit workspace width
Teacher service		
Teacher cache reads	Enabled	Reuse fixed-teacher prefixes
Teacher chunked prefill	Disabled	Preserve complete pooling requests

Note. Not the historical run configuration. The hybrid-teacher probe in Appendix D overrides the prefill setting.

Pool capacity controls resident trajectories; the data-loader batch controls admission. For data-parallel degree D_{DP} and optimizer minibatches n_{mb}, require B divisible by $D_{\mathrm{DP}} n_{\mathrm{mb}}$. Strict mode uses one minibatch/replay epoch and permits gradient accumulation. Dispatch only divisible subsets at epoch boundaries; excluded entries generate nothing. Finite budgets may leave unfinished trajectories, not untrained generated suffixes deferred across an update.

Empty rows carry zero loss. The success/skip, retirement, and finite-budget semantics are specified in Section 3.1.

A.2 Teacher Pooling, Window Alignment, and Transfer

Token-embedding pooling disables output activation and additional normalization, returning final states for the original frozen LM head held by the learner. Hidden widths may differ; token IDs and positions must align. FSDP keeps one projection per configured teacher; the Megatron adapter supports one teacher.

With an extended prompt of P tokens and suffix length m, a reply covering the last r positions starts at $o = P+m-r$. The predictive window $[P-1, P+m-1)$ has local bounds $[P-1-o, P+m-1-o)$, requiring $o \leq P-1$. If $o > P-1$, prefix caching has hidden a required boundary state. The request is retried without cache reads; bounds are checked before target pairing. The prefill budget covers the full visible sequence. Production pooling disables chunked prefill; the hybrid compatibility probe is labelled separately.

Replies enter a pinned BF16 CPU cache. Five `int64` receipt fields (model slot, endpoint, ticket, window start/stop) cost 40 bytes per sample, excluding protocol metadata. The slot selects the head; endpoint/ticket select the cached tensor. NCCL sends only the window, after shape/dtype/order checks. Tensor/context-parallel groups redeem shared rows once and broadcast them, avoiding destructive duplicate redemption.

States are scattered to packed predictive positions; loss selection discards rows outside the response window. Thus N in Equation 5 counts evaluated rows, potentially more than supervised rows. Packing must preserve sequence boundaries and reset recurrent/linear-attention state.

A.3 Cache and Resource Accounting

Section 3.3 accounts for repeated teacher positions and student prefill. Actual cache misses, evictions, and boundary retries must be measured rather than inferred from returned row counts.

Teacher and student serving use disjoint GPUs, possibly on one node. NCCL rejects a sender and consumer on the same physical device. Hidden-state payloads can exceed top-k payloads, and the resident teacher head remains part of learner memory accounting.

B Kernel Derivation and Numerical Validation

B.1 Aggregation and Distributed Normalization

One-use coverage does not preserve response-mean weighting or update cadence. If data-parallel rank r owns fresh rows $\mathcal{N}_{k,r}$, the required global token-mean gradient is

$$\nabla_\theta \widehat{\mathcal{L}}^{(k)}_{\text{Flash}} = \frac{1}{Z_k} \sum_{r=1}^{D_{\text{DP}}} \sum_{(i,t)\in\mathcal{N}_{k,r}} \nabla_\theta \ell_{\text{RKL}}(\theta; s_{i,t}), \qquad Z_k = \sum_r |\mathcal{N}_{k,r}|. \tag{6}$$

If the backend averages rank gradients, each local numerator must be scaled by D_{DP}/Z_k before that average, with corresponding handling of microbatch accumulation. Empty rows contribute zero, not a separate local mean. Tensor/context-parallel replicas of the same logical loss row are not additional data-parallel observations.

B.2 KL Kernel Derivation

Full-support loss statistics. For one state, let $a_v = z_{\text{S},v}/\tau$ and $b_v = z_{\text{T},v}/\tau$ with linear heads $z_M = W_M h_M$. Define $L_{\text{S}} = \text{LSE}_v\, a_v$, $L_{\text{T}} = \text{LSE}_v\, b_v$, $p_v = e^{a_v - L_{\text{S}}}$, and $q_v = e^{b_v - L_{\text{T}}}$. The two normalizers and two student-weighted moments give

$$E_{ss} = \sum_v p_v a_v, \qquad E_{st} = \sum_v p_v b_v, \qquad \ell_{\text{RKL}} = E_{ss} - E_{st} - L_{\text{S}} + L_{\text{T}}. \tag{7}$$

Proposition 1 (Dense-objective equivalence)**.** *For finite supplied states and linear heads, aligned positions and vocabulary, and a frozen teacher, tiling evaluates $\sum_n w_n D_{\text{KL}}(p_n \| q_n)$ and its student hidden-state/head derivatives exactly in real arithmetic. Here p_n, q_n are the two softmax distributions at row n, and w_n is its fixed mask/normalization weight. Both passes cover the same partition.*

Online sufficient statistics. For a vocabulary tile $\mathcal{V}_b$, the student-side recurrence is

$$\begin{aligned} m' &= \max\{m, \max_{v\in\mathcal{V}_b} a_v\}, \qquad r = e^{m-m'}, \\ \ell' &= r\ell + \sum_{v\in\mathcal{V}_b} e^{a_v - m'}, \\ u'_{ss} &= r u_{ss} + \sum_{v\in\mathcal{V}_b} e^{a_v - m'} a_v, \\ u'_{st} &= r u_{st} + \sum_{v\in\mathcal{V}_b} e^{a_v - m'} b_v. \end{aligned} \tag{8}$$

Initialize $m = -\infty$ and $\ell = u_{ss} = u_{st} = 0$. After the final tile, $L_{\text{S}} = m + \log \ell$, $E_{ss} = u_{ss}/\ell$, and $E_{st} = u_{st}/\ell$; a separate maximum and exponential sum give L_{T}. These statistics include every vocabulary item. Empty or padding-only tiles are skipped, avoiding an undefined $-\infty - (-\infty)$ when updating the initial maximum.

Analytic derivatives. For one row, set $d_v = \log p_v - \log q_v$. Since the teacher is frozen, the softmax Jacobian gives

$$\begin{aligned}\frac{\partial \ell_{\mathrm{RKL}}}{\partial a_v} &= \sum_j p_j(\mathbb{I}\{j=v\} - p_v)(d_j + 1) \\ &= p_v\left(d_v - \sum_j p_j d_j\right) \\ &= p_v\big[(a_v - b_v) - (E_{ss} - E_{st})\big]. \end{aligned} \tag{9}$$

The normalizer difference cancels from the centered term. Dividing by τ converts this derivative to unscaled logits. A masked, weighted batch loss multiplies the row gradient by its upstream scalar before the projection gradients in Equation 10 are accumulated.

For a token chunk with student states X and tile gradient G_b with respect to unscaled logits, the projection gradients are

$$\nabla_X \mathcal{L} \mathrel{+}= G_b W_{\mathrm{S}}[\mathcal{V}_b, :], \qquad \nabla_{W_{\mathrm{S}}[\mathcal{V}_b,:]} \mathcal{L} \mathrel{+}= G_b^\top X. \tag{10}$$

Only student states and head weights receive these gradients.

Proof of Proposition 1. The rescaling in Equation 8 preserves the exponential and logit-weighted sums over all processed tiles. Induction over the vocabulary partition recovers the full normalizers and moments, hence the dense RKL in Equation 7. The exact derivative in Equation 9, fixed row weights, and linear adjoints in Equation 10 recover both student gradients. Floating-point reassociation and precision are separate checks. □

Forward KL uses teacher-weighted moments and the logit derivative $p_v - q_v$. Fixed-mixture JSD first computes both normalizers, then an additional vocabulary pass for mixture statistics. Top-k variants change support and are separate objectives. The reported RKL path saves four scalar statistics per token and recomputes projections; resident heads and dense head gradients remain costs.

Combining vocabulary shards. For rank-local student maxima m_r, exponential sums l_r, and unnormalized moments $u_{ss,r}, u_{st,r}$, global statistics are recovered by

$$m = \max_r m_r, \qquad (l, u_{ss}, u_{st}) = \sum_r e^{m_r - m}(l_r, u_{ss,r}, u_{st,r}). \tag{11}$$

Then $L_{\mathrm{S}} = m + \log l$ and $E_{ss} = u_{ss}/l$, $E_{st} = u_{st}/l$. Teacher maxima and sums are combined analogously. Only scalar statistics are exchanged, but hidden-gradient collectives remain. Padding is excluded; empty shards contribute a zero tuple with $m_r = -\infty$. Each row requires at least one finite valid logit across shards.

Library GEMMs produce tiles; Triton fuses reductions and logit-gradient bodies (Tillet et al., 2019). FSDP aggregates data-parallel gradients, while Megatron ranks own matching vocabulary shards with local head gradients and tensor/sequence-parallel hidden-gradient collectives. Recomputed projections retain $\mathcal{O}(NV(d_{\mathrm{S}} + d_{\mathrm{T}}))$ arithmetic. BF16 opt-in policies and their eligible paths are specified in Appendix D.

B.3 Validation and Measurement Scope

Numerical checks. Small-tensor tests compare with dense RKL, including unequal hidden widths, self-distillation, uneven tiles, forced Triton dispatch, distributed statistics, and vocabulary padding. Relative/absolute tolerances are $2 \times 10^{-4}/2 \times 10^{-5}$ for loss and sampled-token log probabilities, and $2 \times 10^{-3}/2 \times 10^{-5}$ for student hidden/head gradients. These are test criteria, not measured large-model errors. Same-input checks isolate tiling; native-teacher checks additionally test pooling, cache, transport casting, and head reconstruction at aligned positions.

Serving and transport checks. State-machine tests check loss spans and response limits; controlled-delay scheduling tests check ordering, not GPU throughput. Window tests exercise boundary misses; transport checks cover dtype, receipt identity, and single redemption for shared rows.

Quality and cost. Budget-matched speedup compares elapsed time at the same supervised-token budget and total accelerator allocation, including teacher and idle devices without double-counting shared roles. This differs from time to a quality target. A fixed prefix bank measures loss on common states, whereas task scores assess completed responses. An unreached target gives a censored time-to-target observation, not an extrapolated crossing. Appendix D specifies the separate fixed-state component protocol.

C Implementation-Level Comparisons

Versioned framework paths. TRL's GKD implementation already supports full-distribution OPD (Hugging Face, 2025). Its synchronous `DistillationTrainer` checkpoints token chunks while projecting $C \times V$ logits (Hugging Face, 2026b). The examined asynchronous trainer instead queues returned completions and uses reduced teacher information: teacher candidates for FKL, the sampled token for pure RKL, and that token plus teacher top-1 for mixed JSD (Hugging Face, 2026a). An optional tail bucket retains residual mass, not within-tail structure. The currently documented `verl` path already overlaps per-sample teacher calls with other rollouts (Helwig, 2026); our dispatch builds on that capability.

Loss implementation rather than a new primitive. Liger's fused-linear JSD/KL accepts hidden states and heads with different widths (Liger-Kernel Contributors, 2026). In the examined version, each token chunk projects the full vocabulary and computes gradients during forward; total forward-plus-backward time is therefore the comparable metric. Flash forms vocabulary tiles before projection and recomputes them during backward. Its $\mathcal{O}(CB_v)$ workspace does not establish dominance over independently tuned token chunks. Nitrobrew already combines hidden-state transport, local teacher-head reconstruction, online tiled divergence, and analytic gradients (Pai et al., 2026). Its technical post does not specify the fresh-suffix consume-before-continuation contract, but this is not evidence that its code cannot be adapted. Optional reduced-rank compression is a separate approximate path, and no Flash-versus-Nitrobrew performance comparison is reported here.

D B200 Component Measurements

This appendix supplies reproducibility details and exact values for the service, delivery, and learner-backend plots in Figures 4–6. The study uses an eight-B200 machine; individual benchmarks use one, two, or four devices as specified. These fixed-input measurements are separate from the historical trace and the downstream task comparison.

D.1 Inputs, Precision, and Measurement Protocol

Fixed state bank. The primary pair is Qwen3.5-4B (student) and Qwen3.6-35B-A3B (teacher), with hidden widths 2,560 and 2,048 and vocabulary 248,320. The complete token-to-ID mappings match. We select the first 32 eligible examples from a locally available reasoning corpus, with prompt length at most 1,024 and at least 1,024 response tokens, after text-message parsing, content-hash deduplication, and chat-template application. From each stored response we capture 1,024 predictive positions using the respective Hugging Face model in BF16, yielding 32,768 states. These are teacher-forced corpus states, not student on-policy samples. Subsets of 1,024 and 8,192 rows are prefixes of the same bank, so the sizes are not independent data samples.

The benchmark objective is unclipped, full-support reverse KL, temperature one, averaged uniformly over evaluated tokens. The teacher states and head are frozen; both student hidden states and the student head require gradients. This includes the head-gradient cost even though the source student ties its input and output embeddings; the benchmark does not execute the decoder or propagate a gradient through its embedding lookup. Initial state capture used the image's default TF32 policy. The stored BF16 states are held identical across all kernel comparisons; strict kernel arithmetic does not retroactively make the state-capture forward pass TF32-free.

Software and hardware. Each NVIDIA B200 reports 183,359 MiB memory. The environment uses PyTorch 2.11.0a0+eb65b36914.nv26.2, Transformers 5.8.1, Triton 3.7.1, Liger-kernel 0.8.3, and

Ray 2.53.0. The imported vLLM source reports 0.20.2rc1.dev388+g2e1b8ecc5.d20260805; installed distribution metadata instead reports 0.18.0+gf4964c907.d20260726.cu131. The runtime source, not that metadata alone, determines the teacher configuration behavior. Measurements use disjoint device allocations when independent component jobs run concurrently; host resources are shared, so these are not exclusive-node benchmark runs.

References and tuning. Dense uses BF16 linear projections and FP32 log-softmax. The compiled chunked reference checkpoints token chunks and compiles the full-vocabulary loss; it is a custom implementation, not a claim to benchmark a particular TRL release. Stock Liger and a separately named variant that casts logits to FP32 before its loss body are measured. Liger may compute gradients during its forward call, so all headline timings include both forward and backward. Chunked references search token chunks in $\{256, 1024, 4096, 8192\}$. Flash searches the coupled (C, B_v) pairs $(1024, 8192)$, $(2048, 16384)$, $(4096, 32768)$, $(8192, 65536)$; this is a bounded search, not an exhaustive Cartesian grid. Three warmups and five timed iterations select the fastest configuration for each implementation and size. Selection timings are not pooled with the final measurements. The original search followed a backend-level 64-row numerical gate, not a separate dense check of every tuning shape. The two opt-in candidates were added after observing the original bottleneck; they are post-hoc optimizations rather than preregistered hypotheses.

Each selected kernel receives 20 warmups and three consecutive rounds of 100 iterations in one process. We report the median wall time, with CUDA events retained separately. Percentiles describe iteration variability, not a confidence interval over independent runs or training seeds. Peak allocated and reserved memory are measured in a separate process after one warmup, gradient clearing, garbage collection, allocator-cache clearing, and peak reset. Reported absolute peaks include resident states and heads, but exclude model decoder weights and optimizer state. Memory differences between precision groups can reflect different selected chunks.

D.2 Kernel Precision and Memory–Latency Trade-offs

The compatibility layer disables TF32 to protect FP32 operations such as long-position rotary calculations. Original backward therefore uses FP32 gradient GEMMs. A warmed $N = 8{,}192$ profile records 329.37 of 394.41 ms GPU activity in SIMT SGEMM gradient kernels, motivating a local precision change rather than relaxing the global policy.

We evaluate two default-off candidates. **BF16-grad** rounds the analytic logit-gradient operand to BF16 for gradient GEMMs, with FP32 GEMM outputs and tile accumulation. **BF16-IO** additionally retains the already-BF16 projection result in BF16 storage at temperature one and stores logit gradients in BF16; Triton explicitly promotes loaded logits to FP32 before normalization and KL arithmetic. The storage option is restricted to the eligible Triton/BF16 path and paired with BF16 gradient GEMMs. It does not change the global TF32 setting. These candidates keep full vocabulary support but introduce finite-precision gradient rounding; algebraic full-support equivalence is not a bitwise-equality claim.

At 32K rows, original Flash's 5th–95th iteration percentiles are 1,575.55–1,579.47 ms; BF16-IO's are 198.47–212.03 ms. A separate TF32-enabled ablation lowers original Flash to 383.60 ms, but changes the production precision policy and is not a general fix.

Numerical gates. The dense reference uses identical BF16 inputs and FP32 log-softmax. The original matrix checks 64 rows; its early records do not identify the FP32 matmul policy and therefore cannot certify every timed precision/shape. Each candidate additionally checks $N \in \{1024, 8192, 32768\}$ with its selected $(8192, 65536)$ tiles. Acceptance thresholds are absolute and relative loss error at most 0.01 and relative L_2 error at most 0.02 for each student gradient. These normwise study gates are distinct from the small-tensor elementwise rtol/atol regression criteria in Appendix B.3. Across the six candidate checks, the maximum loss relative error is below 3.2×10^{-6} and both gradient relative errors are below 0.0032. Table 4 reports every tested candidate/size combination. These are fixed-state observations, not error bounds for arbitrary trained trajectories. A separate 62-test regression suite covers default FP32/FP64 behavior, mixed head dtypes, RKL/FKL/JSD, temperatures $0.7, 1, 1.5$, uneven tiles, and both storage modes. It is not a substitute for an end-task evaluation of the candidates.

Table 3: **Single-device memory–latency trade-off.** Full-vocabulary RKL at 32,768 states on one B200, with TF32 disabled.

Head-loss implementation	F+B time (ms)↓	Peak allocation (GiB)↓
Reference implementations		
Dense	344.31	153.99
Compiled chunked	261.15	18.93
Liger	**171.26**	22.67
Liger FP32 logits	172.57	22.67
Flash kernel policies		
Flash original	1,577.24	15.18
Flash BF16-grad	335.00	15.18
Flash BF16-IO	206.08	**10.18**

Note. Latency-selected configurations; bold marks column minima, not an overall winner. BF16 policies are opt-in. Peak includes states/heads but excludes the decoder and optimizer.

Table 4: **Numerical agreement at the tested sizes.** Opt-in candidates compared with the identical-input dense RKL reference.

		Loss	Gradient relative L_2 error	
N	**Precision policy**	Relative error	h_{S}	W_{S}
1,024	BF16-grad	2.02×10^{-6}	2.50×10^{-4}	1.83×10^{-4}
	BF16-IO	3.03×10^{-6}	2.62×10^{-4}	1.63×10^{-4}
8,192	BF16-grad	3.17×10^{-6}	3.40×10^{-4}	2.28×10^{-4}
	BF16-IO	3.17×10^{-6}	3.67×10^{-4}	2.50×10^{-4}
32,768	BF16-grad	2.72×10^{-6}	3.85×10^{-4}	3.12×10^{-3}
	BF16-IO	2.72×10^{-6}	3.82×10^{-4}	3.12×10^{-3}

Note. $C = 8192$, $B_v = 65536$, $\tau = 1$. Fixed-state observations, not error bounds on trained trajectories.

D.3 Vocabulary-Parallel Kernel

Four NCCL ranks shard the vocabulary, replicate hidden inputs, and include statistic reductions and the full hidden-gradient all-reduce. This differs from a complete sequence-parallel training stack that may reduce-scatter upstream. We fix $C = 8192$, $B_v = 32768$, disable TF32, and use 20 warmups plus 300 measured iterations per policy and size. A barrier precedes each measurement and is excluded from its timer. The reported latency is the median of the per-iteration maximum rank wall time, not the maximum of rank medians. Memory is the largest per-rank peak from a separate run. All four ranks pass the dense-reference loss and gradient checks at $N = 1024$ for all three policies.

The per-rank memory comparison at 32K is 6.26 to 3.86 GiB. This same-device-count comparison cannot be ranked directly against one-B200 Liger timings.

D.4 Serial Teacher Replay and Cache Behavior

For each teacher, four fixed corpus trajectories of 1,024 response tokens are replayed at $H \in \{128, 256, 512, 1024\}$, with three repeats and both cache policies: 360 requests per teacher. Cache-on/off order alternates, with a reset before each trajectory and warmup for both policies. Timings sum sequential requests including output CPU conversion, excluding initialization and student decoding. Untimed no-cache replies supply window references. Missing boundary states trigger no-cache retry; none occurred in these cases, which does not validate every boundary residue.

Table 5: **Vocabulary-parallel head loss.** Precision-policy comparison on the same four devices, including NCCL reductions.

	8,192 states		32,768 states	
Flash kernel policy	F+B (ms)↓	Peak/rank (GiB)↓	F+B (ms)↓	Peak/rank (GiB)↓
Original	97.63	5.63	391.45	6.26
BF16-grad	21.34	5.63	86.43	6.26
BF16-IO	**13.27**	**3.24**	**54.27**	**3.86**

Note. Time is the median of per-iteration rank maxima; memory is the largest per-rank peak. Bold marks column minima. This is not a full-learner scaling result or a TP4 Liger comparison.

Teacher-serving configuration. The imported vLLM version requires chunked prefill for the primary hybrid teacher's prefix-cache alignment mode. Its measurements therefore use an explicitly labelled compatibility probe with chunked prefill enabled, unlike the production default. The probe uses one active request shorter than the 4,096-token budget, TP1, eager execution, and unnormalized token embeddings; it does not validate batched production pooling. The engine logs resolve a 1,056-token attention block after hybrid-state page alignment, larger than the requested base block size of 16. This limits reuse at these short horizons.

The secondary Qwen3-4B teacher uses the standard no-chunked-prefill setting and its own tokenizer-aligned corpus subset. It is not directly comparable in absolute latency to the primary model. Cache-on/off comparisons within each model hold its token sequences fixed.

Table 6: **Teacher replay across suffix horizons.** Cache off/on report median serial service time per fixed 1,024-token response.

		Trajectory time (ms)		Cache-on rows	Cache error
	H	Cache off	Cache on	Returned/sent	Max. rel. L_2
Qwen3.6-35B-A3B	*Compatibility probe*				
	128	1,020.89	837.44	0.672	0.0000
	256	558.33	466.61	0.692	0.0000
	512	277.35	278.70	0.890	0.0000
	1,024	184.03	183.46	1.000	0.0000
Qwen3-4B	*No chunked prefill*				
	128	161.04	153.54	0.190	0.0300
	256	81.52	78.27	0.348	0.0272
	512	39.29	38.62	0.613	0.0184
	1,024	20.34	20.36	1.000	0.0000

Note. Cache-on row counts are not FLOPs. Error is cache-on versus no-cache. The Qwen3-4B $H = 128$ observations retain the 11.54 s outlier.

The primary probe's cached windows match its no-cache BF16 references in these cases. Qwen3-4B's maximum relative L_2 difference is 0.0300, with unmeasured logit/KL/gradient impact. Reduced returned rows are not a direct measure of executed work or latency savings.

The median does not characterize the synchronization tail. For Qwen3-4B at $H = 128$ with caching, the 12 observed trajectory times have a 95th percentile of 5,278.16 ms and a maximum of 11,535.15 ms, despite the 153.54 ms median. A retained 11,391.91 ms request in the first trajectory/repeat accounts for most of the outlier. Without request-level profiling, compilation, cache initialization, and host interference remain unresolved explanations. Short-request warmup need not cover every cache-hit branch, and 12 observations do not establish a population-tail estimate. Cache-window agreement also does not establish native HF/vLLM logit agreement.

D.5 Payload Transfer and Staging Costs

Two GPU processes compare resident NCCL with FP32-source staging through a BF16 pinned-CPU receipt cache, selected-window gather/H2D, fingerprint, and NCCL. Teacher width is 2,048; the largest payload repeats the same bank to test transfer size. Each size/path uses five warmups and 20 measurements, excluding lazy channel initialization and the pre-timing barrier. All 160 payloads pass untimed CPU bitwise checks. Ray RPC and directory/control-plane latency are excluded. The source is captured BF16 upcast to FP32, so this validates wire integrity, not lossless native-FP32 quantization. Maximum endpoint duration is not a globally timestamped cross-process critical path.

Table 7: **Payload delivery: resident versus staged paths.** BF16 payloads transferred between two GPU processes.

Payload (MiB)	Resident NCCL		Staged + NCCL	
	Time (ms)	Cache put (ms)	Time (ms)	Cache put (ms)
4	0.073	–	0.878	0.139
32	0.125	–	13.841	0.657
128	0.368	–	55.056	2.473
512	0.937	–	226.574	9.705

Note. Time is the median maximum endpoint duration; "–": no cache insertion. Cache put is only a substage. Staged includes gather/H2D, fingerprint, and NCCL, not Ray RPC.

Gather/H2D includes CPU gathering, allocation, and copying; it is not an isolated link-bandwidth measurement. Separately measured substage medians need not sum to the end-to-end median. Figure 5 plots the corresponding endpoint-duration comparison.

D.6 Measurement Records and Scope

Records cover 36 single-device configurations (10,800 iterations, including TF32 ablation), six TP4 configurations (1,800 matched iterations), 720 teacher requests, and 160 transfers. Raw timings, selected configurations, separate memory results, numerical checks, and logs are preserved. Final source hashes are not per-run immutable snapshots. Another 32 TP/default-precision and scheduling tests establish correctness/ordering, not throughput.

Integrated training attempts with the original kernel failed during runtime/transport initialization, before any optimizer update; they contribute no training timing result. Component gains do not establish quality-equivalent training speedups. These experiments do not attribute the training results in Table 1 to individual components or validate the BF16 policies' downstream learning behavior. No Nitrobrew timing comparison is included.