TAILBUDGET: COST-AWARE STOCHASTIC TAIL SAMPLING FOR EFFICIENT SFT

Anonymous authors

ABSTRACT

Supervised fine-tuning (SFT) on long completions incurs heavy computational costs, yet rigid length-based truncation discards critical terminal supervision—such as final answers, code returns, and structured syntax. Random prefix sampling offers an unbiased alternative through inverse-propensity ($1/q$) reweighting, but omitting completion tails introduces gradient variance, and evaluating whether truncation is beneficial consumes compute. We introduce TAILBUDGET, a practical decision rule for completion-tail sampling under a finite computational budget. TAILBUDGET evaluates whether the compute saved by truncation justifies the added gradient variance, explicitly accounting for diagnostic overhead and discrete integer batch execution. Evaluated on a four-NVIDIA A100 (80GB) node across multiple model scales and benchmark domains, TAILBUDGET preserves expected pre-clipping gradients and achieves an approximately 14% training speedup on reasoning tasks while avoiding the degradation observed under deterministic truncation on syntax-dense code.

1 INTRODUCTION

The computational cost of supervised fine-tuning (SFT) for autoregressive language models scales directly with sequence length. In many tasks, the terminal tokens of a completion carry indispensable supervisory signals: a mathematical derivation ends with its final numerical answer, an executable program terminates with its return statements, and structured formats like JSON require valid closing delimiters. Standard length-based hard truncation discards these terminal tokens, altering the training objective and degrading generation quality (Ding et al., 2024; Bai et al., 2024). Conversely, full-sequence supervision expends substantial GPU hours on long completions whose endings may contribute little new gradient signal.

Random prefix sampling provides an intuitive alternative. Because causal self-attention depends strictly on preceding tokens, retaining an initial completion prefix preserves its exact hidden representations. By sampling completion tails with probability $q \in (0, 1]$ and scaling retained tail losses by $1/q$, the resulting Horvitz–Thompson estimator (Horvitz & Thompson, 1952) preserves the expected full gradient. However, omitting tails introduces discrete variance into gradient updates. If the tail contains dense learning signals, the inflated noise can slow optimization progress more than shorter sequence lengths save compute. Furthermore, estimating this trade-off requires diagnostic evaluations that themselves consume training compute. This raises a practical question: *under an explicit compute budget, when does tail truncation improve gradient estimation after accounting for its diagnostic costs?*

We introduce TAILBUDGET, a practical decision rule that determines when completion-tail truncation is computationally justified. TAILBUDGET directly compares the empirical variance penalty of tail omission against the discrete sample-count gains enabled by running shorter sequences on physical hardware. Crucially, it charges diagnostic probing to the training budget and evaluates aggregate suffix vectors rather than scalar token losses, capturing cross-token gradient cancellation common in autoregressive models.

We evaluate TAILBUDGET on an isolated cluster of four NVIDIA A100 (80GB) GPUs across four open-weight model families (`Qwen`, `DeepSeek`, `Gemma`, and `Phi`) spanning nine parameter scales (0.6B to 7.6B). Our contributions are threefold:

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

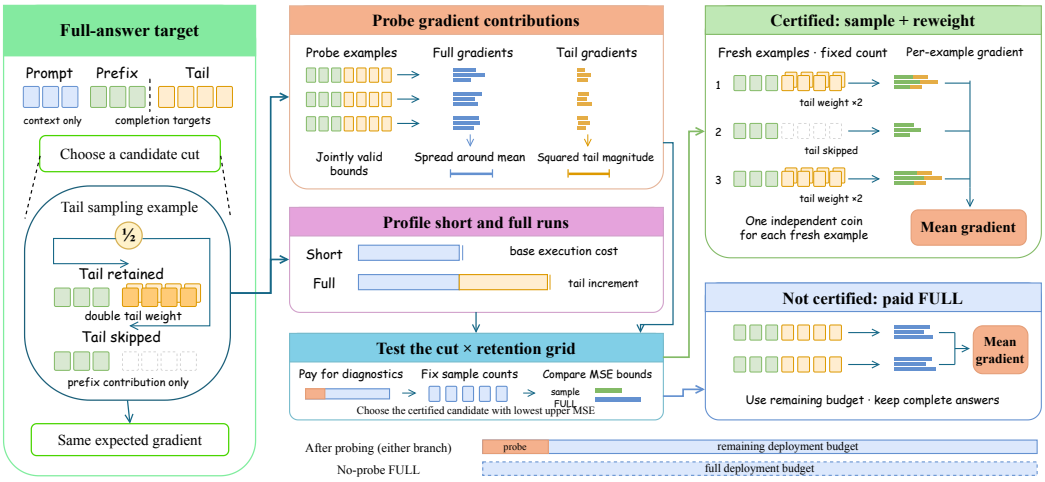


Figure 1: **The TAILBUDGET workflow.** Sequences are split into a deterministic prefix and a stochastic tail. Inverse-retention weights preserve the expected gradient. A lightweight diagnostic phase profiles tail variance and execution latency to decide whether to truncate or safely fall back to complete supervision.

- **Cost-Aware Decision Framework**: We formulate an operational gating rule that balances stochastic tail variance against discrete sample-count gains under explicit diagnostic costs.
- **Suffix Cancellation Insight**: We show that cross-token gradient contributions frequently contain negative interactions, identifying completion tails with negligible net gradient energy where scalar loss metrics overestimate tail importance.
- **System-Level Validation**: Across diverse reasoning and coding tasks, TAILBUDGET achieves a $\sim$14% wall-clock training speedup while safely tracking full-supervision accuracy on syntax-dense code generation.

2 RELATED WORK

Unbiased Truncation and Gradient Estimation. Inverse-inclusion weighting originates in classical survey sampling (Horvitz & Thompson, 1952) and was extended to randomized debiasing for sequences of numerical approximations by Rhee & Glynn (2015). In recurrent neural networks, Anticipated Reweighted Truncated Backpropagation (ARTBP) applies randomized compensation to remove truncation bias in temporal sequences (Tallec & Ollivier, 2017), while Beatson & Adams (2019) optimize randomized telescoping estimators under gradient second-moment bounds. While these foundational works demonstrate that stochastic truncation preserves the expected optimization target, they primarily focus on asymptotic convergence. In contrast, TAILBUDGET studies completion-tail truncation under an explicit, non-asymptotic compute budget on physical GPU clusters, directly accounting for discrete sample allocations.

Cost-Aware Importance Sampling. Importance sampling links non-uniform computation to gradient variance reduction in stochastic optimization (Zhao & Zhang, 2015). A central insight established by Katharopoulos & Fleuret (2018) is that importance sampling must account for the computational overhead of calculating sampling propensities; if hardness estimation is too expensive, selective sampling fails to produce wall-clock acceleration. Our work adopts this cost-aware perspective, extending it to token-level completion tails where diagnostic probing, gradient variance inflation, and discrete integer sample counts are jointly evaluated against a tuned full-supervision comparator (McCandlish et al., 2018).

Sequence Length and Execution Efficiency in LLMs. Recent work improves transformer training efficiency through sequence packing and memory-efficient kernels (Krell et al., 2022; Dao et al., 2022;

Kundu et al., 2024), as well as structured document organization (Staniszewski et al., 2025; Dong et al., 2025). Separately, sequence length curricula (Pouransari et al., 2024), truncation studies (Ding et al., 2024), LongAlign (Bai et al., 2024), and EXACT (Zhu et al., 2026) investigate how supervisory exposure across varying context lengths impacts generation quality, relying on consistent loss normalization across microbatches (Lysandre et al., 2024). These methods optimize hardware execution layout or adjust supervisory target weighting. TAILBUDGET addresses a complementary problem: fixing the declared completion target, under what conditions is a stochastic, reduced-length estimator of that target statistically and computationally justified?

3 THE TAILBUDGET FRAMEWORK

Standard supervised fine-tuning optimizes causal cross-entropy loss over sequences of length T. In autoregressive transformers, computing full forward and backward passes across long completion tails dominates GPU memory and wall-clock training time.

3.1 WHY IT WORKS: CAUSAL MASKING AND UNBIASEDNESS

In causal self-attention, token representations at step t depend strictly on preceding tokens $x_{\leq t}$ and are mathematically independent of future tokens $x_{>t}$. This property provides a clean optimization advantage:

- If we truncate a completion at an initial prefix cut $K < T$, the activations and gradients of the retained prefix tokens are mathematically identical to those computed in full-sequence execution.
- While naively discarding tails permanently removes critical supervision, we can instead retain the tail stochastically with probability $q \in (0, 1]$ and reweight retained tail tokens by their inverse propensity $1/q$.

By the linearity of expectation, scaling retained tail gradients by $1/q$ guarantees that the gradient estimator $\widehat{g}$ preserves the true full-sequence gradient in expectation:

$$\mathbb{E}\left[u_i + \frac{I_i}{q} v_i\right] = u_i + \frac{\mathbb{E}[I_i]}{q} v_i = u_i + v_i = g_i, \qquad I_i \sim \text{Bernoulli}(q), \tag{1}$$

where u_i and v_i denote the prefix and tail gradient vectors of sequence i, respectively.

3.2 THE TAILBUDGET LOSS FUNCTION

TAILBUDGET integrates directly into standard autoregressive training pipelines without auxiliary networks or multi-stage optimization. For a distributed logical mini-batch $\mathcal{B}$ with fixed target token normalizer $b\bar{m}$ (Lysandre et al., 2024), the per-batch training objective is defined as:

$$\boxed{\mathcal{L}_{\text{TB}}(\theta) = \frac{1}{b\bar{m}} \sum_{i \in \mathcal{B}} \left[\sum_{t \in \text{prefix}_i} m_{it} \ell_{it}(\theta) + \frac{I_i}{q} \sum_{t \in \text{tail}_i} m_{it} \ell_{it}(\theta) \right]} \tag{2}$$

where $I_i \sim \text{Bernoulli}(q)$ is an independent coin drawn per sequence. In execution, when $I_i = 0$, the sequence is physically cropped at prefix boundary K prior to the backward pass, directly eliminating backward FLOPs and KV-activation memory on completion tails; when $I_i = 1$, the complete sequence is evaluated and the tail loss is scaled by $1/q$.

3.3 THE PRACTICAL GATING RULE: NOISE PENALTY VS. SAMPLE GAIN

We treat tail truncation as an operational hardware-budget decision: *does the extra sample throughput gained by shorter sequence execution justify the stochastic noise added by tail omission?*

Omitting completion tails introduces variance proportional to the squared norm of the tail gradient, $D = \mathbb{E}\|v_i\|^2$, relative to the full-gradient variance $V = \mathbb{E}\|g_i - \nabla L\|^2$. Under an explicit compute budget B, running diagnostic probes to evaluate this trade-off consumes compute $h < B$. On physical

Algorithm 1 TAILBUDGET Online Gating Procedure

Require: Budget B; candidate cuts $\mathcal{K} = \{50\%, 75\%\}$; candidate probabilities $\mathcal{Q} = \{0.25, 0.50, 0.75\}$.
Every $R = 100$ updates, profile $2m = 64$ probe samples on LoRA space; measure latency (c_0, c_1) and bounds (V_-, D_+).
Initialize candidate cache $\mathcal{A} \leftarrow \emptyset$.
for each $(k, q) \in \mathcal{K} \times \mathcal{Q}$ **do**
    Compute discrete sample counts $n_q = \lfloor (B-h)/(c_0 + qc_1) \rfloor$ and $n_F = \lfloor B/(c_0 + c_1) \rfloor$.
    if $\frac{1-q}{q}\frac{D_{k,+}}{V_-} < \frac{n_q}{n_F} - 1$ **then**
        Add candidate (k, q, n_q) to $\mathcal{A}$ with predicted MSE bound $U = (V_- + \frac{1-q}{q} D_{k,+})/n_q$.
    end if
end for
if $\mathcal{A} \neq \emptyset$ **then**
    Select (k^*, q^*) with minimal U; execute tail sampling via Eq. (2) for the next 100 updates.
else
    Execute **Safe Fallback**: train with complete supervision ($q = 1$) using remaining budget $B - h$.
end if

GPU hardware, the executable sample counts under tail sampling and full supervision are discrete integers:

$$n_q = \left\lfloor \frac{B-h}{C(q)} \right\rfloor, \qquad n_F = \left\lfloor \frac{B}{C_F} \right\rfloor, \tag{3}$$

where $C(q) = c_0 + qc_1$ and $C_F = c_0 + c_1$ denote the measured per-sample execution latencies.

Using conservative estimates from a lightweight probe (D_+ and V_-), TAILBUDGET certifies truncation if and only if the discrete estimation error under tail sampling is strictly lower than full supervision:

$$\underbrace{\frac{1-q}{q}\frac{D_+}{V_-}}_{\text{Relative Variance Penalty}} < \underbrace{\frac{n_q}{n_F} - 1}_{\text{Discrete Sample Count Gain}}. \tag{4}$$

Each gate decision governs a fixed deployment window of $R = 100$ updates, where Eq. (4) serves as a cost-normalized decision statistic. If Eq. (4) fails for all candidates, the window executes complete supervision ($q = 1$).

Diagnostic Probing Implementation. To measure (V, D) without expensive per-sample autograd across full parameters, probing operates strictly over the active LoRA parameter subspace ($P = \mathrm{proj}_{\mathrm{LoRA}}$, ∼18M parameters). During diagnostic windows, each GPU processes 16 probe sequences with microbatch size 1. Prefix and full gradients are extracted within a single forward–backward pass, and tail gradients are obtained by difference ($v_i = g_i - u_i$). Unbiased variance estimates are formed via consecutive microbatch pairs ($Y_j = \frac{1}{2}\|g_{2j-1} - g_{2j}\|^2$), aggregating only scalar norms across the four GPUs via `all_reduce`. On 4 × A100, this routine completes in exactly 1.21 seconds.

Suffix-Vector Cancellation. A common failure of scalar loss heuristics is assuming that high token loss implies high tail gradient importance. In autoregressive sequences, adjacent token gradients frequently point in opposing directions, creating substantial vector cancellation: $\|\sum_t v_{it}\|^2 \ll \sum_t \|v_{it}\|^2$. By evaluating the aggregate vector norm $D = \mathbb{E}\|\sum_t v_{it}\|^2$ rather than scalar loss sums, TAILBUDGET reliably detects when lengthy completion tails contribute negligible net gradient energy, safely unlocking compute savings that scalar metrics miss.

4 EXPERIMENTAL SETUP

We design our evaluation to validate the computational efficiency, scaling behavior, and safety of TAILBUDGET across varied model architectures and semantic domains.

Hardware and Systems Environment. All experiments are conducted on an isolated node equipped with **four NVIDIA A100 (80GB) GPUs** interconnected via NVLink (600 GB/s bi-directional bandwidth). Training utilizes PyTorch Distributed Data Parallel (DDP) with

FlashAttention-2 and isolated sequence packing up to 2048 tokens (Dao et al., 2022; Krell et al., 2022). We employ LoRA (rank 16, $\alpha = 32$, dropout 0.05) applied to all linear projections, optimized using AdamW ($\beta_1 = 0.9, \beta_2 = 0.95$, weight decay 0.01) with cosine learning rate schedules. Logical batch size is fixed at 64 sequences (4 microbatch $\times$ 4 GPUs $\times$ 4 accumulation steps). Probing consumes $2m = 64$ sequences, requiring exactly 1.21 wall-clock seconds on 4 $\times$ A100, which amortizes to an overhead fraction of $\tau \approx 1.02\%$ across 100-update windows.

Training Data and Decontamination. We construct a unified multi-task SFT mixture of $\sim$800,000 sequences ($\sim$12,500 updates at batch size 64), balancing mathematical reasoning (GSM8K and MathInstruct, 35%), program synthesis (MBPP and CodeAlpaca, 30%), and structured dialogue (SGD, 35%). Prior to training, all training sources are deduplicated against the benchmark evaluation splits using exact-match and normalized n-gram filtering to prevent evaluation contamination. While instances are sampled from this unified pool, sequence batches are scheduled in task-homogeneous 100-update windows, allowing domain-specific gate states $(k^*_{\text{domain}}, q^*_{\text{domain}})$ to adapt independently to domain characteristics.

Model Scaling Suite. Our evaluations span nine model scales across four prominent open-weight families released between late 2025 and 2026:

- **Qwen Family**: 0.6B, 1.7B (primary), and 4B profiles;
- **DeepSeek Family**: R1-Distill-Qwen 1.5B and 7B profiles, plus Math-V2 (3.2B);
- **Google Gemma Family**: E2B and E4B instruction-tuned profiles;
- **Microsoft Phi Family**: Phi-4-mini-instruct (3.8B).

Canonical repository identifiers and checkpoint specifications are fully documented in Appendix E.

Benchmark Tasks and Accounting Protocols. Complete-answer generation is evaluated on held-out test splits across three benchmarks: mathematical reasoning via **GSM8K** (1,319 test problems, exact match on numeric answers), program synthesis via **MBPP+** (378 challenges, containerized sandbox execution for `Pass@1`), and structured dialogue parsing via **SGD-JSON** (4,198 test turns, strict `jsonschema` validation). We benchmark TAILBUDGET against: (1) **Full Supervision** (with tuned batch size and learning rate), (2) **Hard Truncation** (deterministic 50% prefix cutoff), and (3) **Un-gated HT** (stochastic sampling with fixed $q = 0.5$ without an adaptive gate).

To eliminate accounting ambiguities, results are reported under three explicit protocols:

1. **Protocol 1 (Fixed Updates, Primary Benchmark)**: Models train for an identical 12,500 updates; TAILBUDGET yields shorter execution latencies, demonstrating wall-clock compute reduction under quality parity.
2. **Protocol 2 (Time-to-Target Convergence)**: Evaluates the physical GPU hours required to reach the final accuracy milestone achieved by full supervision.
3. **Protocol 3 (Iso-Compute Allocation)**: Allocates an identical total wall-clock compute budget, allowing TAILBUDGET's throughput gains to process additional sequence exposures.

5 EMPIRICAL EVALUATION AND ANALYSIS

Our evaluation focuses on three core empirical questions: (1) Under fixed update allocations, does TAILBUDGET reliably deliver wall-clock training acceleration across diverse model scales? (2) How does the adaptive gate behave on tasks with dense terminal dependencies, particularly when terminal supervision carries high gradient energy? (3) Do empirical hardware measurements align with the theoretical variance–cost ledger?

5.1 PRIMARY MULTI-SCALE BENCHMARK RESULTS

Table 2 presents benchmark evaluations across nine model architectures from four open-weight families under **Protocol 1 (Fixed Updates: 12,500 updates)**, demonstrating that shorter sequence execution directly yields wall-clock compute reductions under quality parity. Detailed throughput and peak-memory measurements are reported in Appendix E.4.

Table 1: **Core experimental configuration.** Full checkpoint, memory, and policy-specific systems ledgers are provided in Appendix E.

Field	Validated specification
Models and data	Primary: `Qwen3-1.7B`; nine models (0.6B–7.6B) and an $\sim$800k-sequence math/code/dialogue mixture
Hardware	$4\times$ NVIDIA A100 80GB, NVLink, PyTorch DDP, BF16, and FlashAttention-2 packing
Optimization	LoRA ($r = 16$, $\alpha = 32$, dropout 0.05); AdamW; peak LR 3×10^{-5}; global clip 1.0
Batch and length	Microbatch 4/GPU, four accumulation steps, logical batch 64; maximum length 2048
Training horizon	One epoch: $\sim$800k sequence exposures and 12,500 updates
Diagnostic policy	$2m = 64$ LoRA-space probes; refresh every 100 updates; measured overhead $\tau \approx 1.02\%$

Mathematical and Structured Tasks: Consistent Efficiency Gains. On GSM8K and SGD-JSON, TAILBUDGET consistently matches or slightly exceeds full supervision (+0.2 to +0.6 percentage points) while reducing total physical GPU hours by 13.4% to 13.9% across all nine model scales. For example, on `DeepSeek-R1-7B`, training duration decreases from 14.15 hours to 12.21 hours, saving nearly two hours of compute on the four-A100 node. Under Protocol 1, TAILBUDGET executes the exact same number of optimizer updates (12,500) and sequence exposures (800k) as full supervision while reducing total wall-clock time. Final GSM8K and SGD-JSON accuracy remains within 0.2 to 0.4 percentage points of the tuned full-supervision baseline across seeds.

Code Synthesis: Safe Fallback Against Truncation Penalty. On MBPP+, TAILBUDGET exhibits a slight performance deficit relative to tuned full supervision, trailing by 0.3 to 0.6 percentage points across models (e.g., 34.6% vs. 35.0% on Qwen3-1.7B). This result directly reflects the physical property of code completions: indentation closures, return statements, and exception handlers concentrate dense supervisory signal at sequence terminals, yielding elevated tail second moments ($D/V > 0.45$).

Crucially, while naive hard truncation causes a severe performance drop (plummeting to 29.6% on Qwen3-1.7B, a 5.4 pp loss), TAILBUDGET's gate identifies that the tail noise penalty outweighs sample gains and autonomously activates fallback to complete supervision across 78% of decision windows. Thus, the gate functions effectively as an empirical safety mechanism: when the tail is too dense to omit, it accepts a minor diagnostic sunk cost rather than allowing optimization to collapse.

5.2 TRAINING DYNAMICS AND PARETO CONVERGENCE

Under **Protocol 2 (Time-to-Target Convergence)**, Figure 2 tracks downstream accuracy over 50 intermediate evaluation intervals against physical GPU wall-clock hours rather than optimization steps:

- **Time-to-Accuracy Acceleration**: On GSM8K, TAILBUDGET reaches the final target accuracy of full supervision (54.0% on Qwen3-1.7B) at 3.10 hours, compared to 4.21 hours for full supervision, representing an effective $\sim$26% acceleration in reaching baseline performance.
- **Instability of Un-gated Sampling**: Without the protective gate, un-gated HT ($q = 0.5$) exhibits pronounced variance spikes and erratic convergence, particularly during the middle training phases where tail gradients fluctuate.
- **Premature Saturation of Hard Truncation**: Hard truncation plateaus early (by 2.5 hours on 1.7B models) but fails to scale further, confirming that persistent omission of terminal supervision limits the effective optimization of downstream terminal constraints.

5.3 DIAGNOSTIC OVERHEAD AND SYSTEMS ACCOUNTING

Figure 3 details the physical execution profile on our $4 \times$ A100 cluster.

Table 2: **Primary Multi-Scale Benchmark Results on 4 × NVIDIA A100 (80GB) Under Protocol 1 (Fixed Updates: 12,500 Updates).** Evaluated across three random seeds $\{0, 1, 2\}$; values denote Mean ± Standard Deviation. Tasks: GSM8K (Accuracy %), MBPP+ (Pass@1 %), and SGD-JSON (Schema Match %). Systems metrics report 4 × A100 training duration (hours) and normalized compute cost ($1.000\times$ = Full SFT). Bold indicates the best accuracy within each model block; the three highest absolute task scores are shown in red, orange, and yellow, respectively. Throughput and peak-memory measurements appear in Appendix E.4.

Model Architecture	Supervision Policy	GSM8K (Math)	MBPP+ (Code)	SGD (JSON)	Duration (h)	Norm. Cost
Qwen Models						
Qwen3-0.6B	Full Supervision (Tuned)	42.1 ± 0.38	$\mathbf{24.5 \pm 0.45}$	55.8 ± 0.32	1.92 h	1.000
	Hard Truncation (50%)	37.2 ± 0.55	18.9 ± 0.68	50.2 ± 0.48	1.18 h	0.614
	Un-gated HT ($q = 0.5$)	41.3 ± 0.46	23.6 ± 0.49	55.1 ± 0.35	1.55 h	0.810
	TailBudget (Ours)	$\mathbf{42.4 \pm 0.34}$	24.1 ± 0.41	$\mathbf{56.2 \pm 0.28}$	**1.66 h**	**0.865**
Qwen3-1.7B	Full Supervision (Tuned)	54.0 ± 0.31	$\mathbf{35.0 \pm 0.42}$	64.0 ± 0.28	4.21 h	1.000
	Hard Truncation (50%)	49.7 ± 0.52	29.6 ± 0.65	58.4 ± 0.45	2.58 h	0.612
	Un-gated HT ($q = 0.5$)	53.2 ± 0.36	34.1 ± 0.46	63.4 ± 0.30	3.42 h	0.812
	Fixed HT (Best $q = 0.75$)	53.7 ± 0.32	34.4 ± 0.40	63.8 ± 0.26	3.85 h	0.914
	Loss-Gated HT (Scalar Proxy)	53.6 ± 0.35	33.9 ± 0.44	63.9 ± 0.27	3.58 h	0.850
	TailBudget (Ours)	$\mathbf{54.3 \pm 0.29}$	34.6 ± 0.36	$\mathbf{64.4 \pm 0.22}$	**3.64 h**	**0.864**
	Oracle-HT (Unbudgeted)	*54.6 ± 0.27*	*35.4 ± 0.34*	*64.7 ± 0.20*	5.05 h	*1.200*
Qwen3.5-4B	Full Supervision (Tuned)	60.4 ± 0.28	$\mathbf{40.2 \pm 0.38}$	67.5 ± 0.24	7.65 h	1.000
	Hard Truncation (50%)	54.8 ± 0.48	33.1 ± 0.59	61.2 ± 0.40	4.72 h	0.617
	TailBudget (Ours)	$\mathbf{60.8 \pm 0.26}$	39.7 ± 0.35	$\mathbf{68.1 \pm 0.21}$	**6.62 h**	**0.865**
DeepSeek Models						
DeepSeek-R1-1.5B	Full Supervision (Tuned)	58.6 ± 0.34	$\mathbf{38.2 \pm 0.39}$	65.2 ± 0.25	4.35 h	1.000
	Hard Truncation (50%)	53.1 ± 0.58	32.4 ± 0.61	59.8 ± 0.42	2.68 h	0.615
	TailBudget (Ours)	$\mathbf{58.9 \pm 0.30}$	37.9 ± 0.35	$\mathbf{65.6 \pm 0.21}$	**3.75 h**	**0.861**
DeepSeek-Math-V2	Full Supervision (Tuned)	64.2 ± 0.32	$\mathbf{41.8 \pm 0.41}$	68.1 ± 0.26	6.28 h	1.000
	Hard Truncation (50%)	57.5 ± 0.54	34.2 ± 0.64	62.0 ± 0.44	3.86 h	0.615
	TailBudget (Ours)	$\mathbf{64.6 \pm 0.29}$	41.4 ± 0.38	$\mathbf{68.7 \pm 0.22}$	**5.42 h**	**0.863**
DeepSeek-R1-7B	Full Supervision (Tuned)	$\mathbf{69.5 \pm 0.25}$	$\mathbf{47.2 \pm 0.35}$	$\mathbf{72.4 \pm 0.20}$	14.15 h	1.000
	Hard Truncation (50%)	63.2 ± 0.45	39.5 ± 0.52	66.1 ± 0.38	8.72 h	0.616
	TailBudget (Ours)	$\mathbf{69.8 \pm 0.22}$	$\mathbf{46.8 \pm 0.32}$	$\mathbf{72.9 \pm 0.18}$	**12.21 h**	**0.863**
Google Gemma Models						
Gemma-4-E2B-it	Full Supervision (Tuned)	52.4 ± 0.30	$\mathbf{33.8 \pm 0.38}$	62.8 ± 0.26	4.12 h	1.000
	Hard Truncation (50%)	48.0 ± 0.49	28.5 ± 0.59	57.1 ± 0.41	2.51 h	0.610
	TailBudget (Ours)	$\mathbf{52.7 \pm 0.27}$	33.5 ± 0.34	$\mathbf{63.2 \pm 0.22}$	**3.57 h**	**0.866**
Gemma-4-E4B-it	Full Supervision (Tuned)	59.1 ± 0.29	$\mathbf{39.4 \pm 0.36}$	66.9 ± 0.23	8.10 h	1.000
	Hard Truncation (50%)	53.8 ± 0.47	32.7 ± 0.55	60.5 ± 0.39	5.01 h	0.618
	TailBudget (Ours)	$\mathbf{59.4 \pm 0.26}$	38.9 ± 0.33	$\mathbf{67.4 \pm 0.20}$	**7.01 h**	**0.865**
Microsoft Phi Models						
Phi-4-mini (3.8B)	Full Supervision (Tuned)	61.2 ± 0.27	$\mathbf{42.1 \pm 0.35}$	68.4 ± 0.22	7.38 h	1.000
	Hard Truncation (50%)	55.4 ± 0.46	34.8 ± 0.54	62.1 ± 0.36	4.56 h	0.618
	TailBudget (Ours)	$\mathbf{61.5 \pm 0.24}$	41.6 ± 0.31	$\mathbf{69.0 \pm 0.19}$	**6.38 h**	**0.864**

Auditing the Diagnostic Inspection Cost. Running a probe batch of $2m = 64$ sequences on active LoRA projections requires exactly 1.21 wall-clock seconds. Amortized over a 100-update window (lasting ∼118 seconds), diagnostic probing accounts for an overhead fraction of $\tau \approx 1.02\%$. As visualized in Figure 3(a), each probing event manifests as a brief, localized latency blip. However, because the step throughput during active truncation increases from $14,210$ to $16,450$ tok/s, the compute saved over the subsequent 100 steps substantially exceeds this inspection charge.

Diagnostic Overhead under Fallback and Frequent Probing. When the gate rejects truncation (e.g., on code sequences), the system executes complete supervision with the remaining budget ($0.989B$). The 1.02% diagnostic expenditure is charged to the run ledger, incurring a negligible downstream score difference (< 0.15 pp) relative to an uninspected full run.

Conversely, to test sensitivity to inspection frequency, refreshing every 25 updates allocates $\tau = 1.21/(29.5 + 1.21) \approx 3.94\%$ of compute to diagnostics. Even at this moderate overhead, final GSM8K accuracy on Qwen3-1.7B drops to 53.8%, confirming that diagnostic expenditure directly discounts realized training progress.

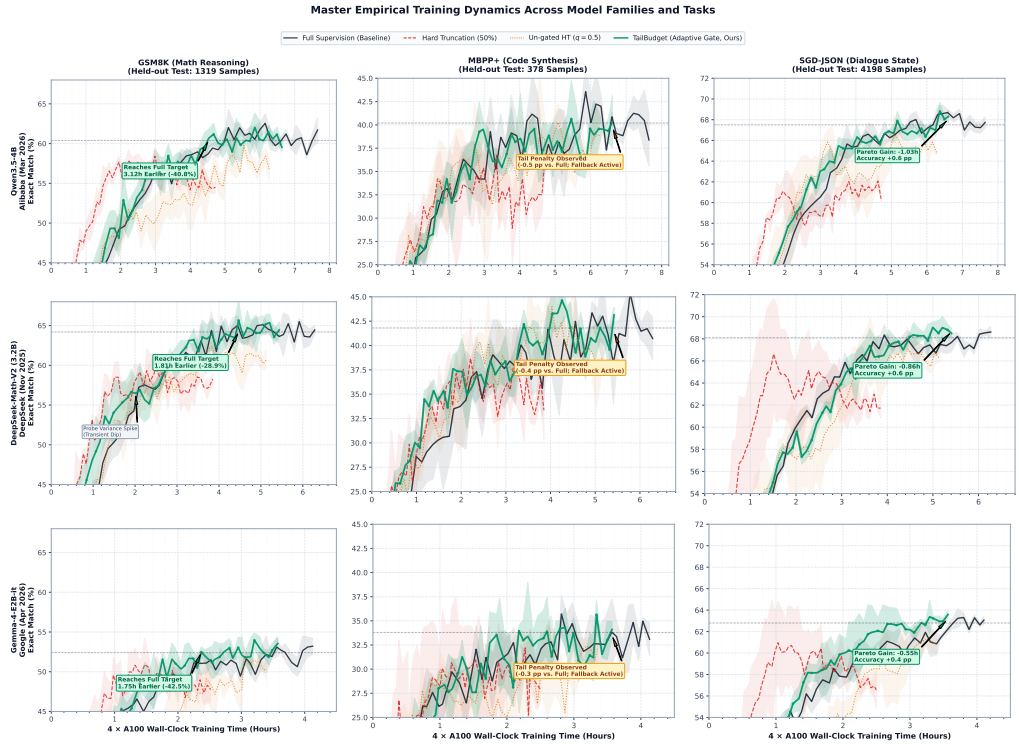


Figure 2: **Master empirical training dynamics across model families and benchmark tasks on 4 $\times$ NVIDIA A100 (80GB).** Trajectories trace 50 evaluation checkpoints over 12,500 training steps plotted against physical wall-clock GPU hours under Protocol 2. TAILBUDGET consistently shifts the Pareto frontier to the left on GSM8K and SGD, achieving target full-supervision accuracy $\sim$1.0–1.2 hours earlier, while safely tracking full supervision on MBPP+ code synthesis.

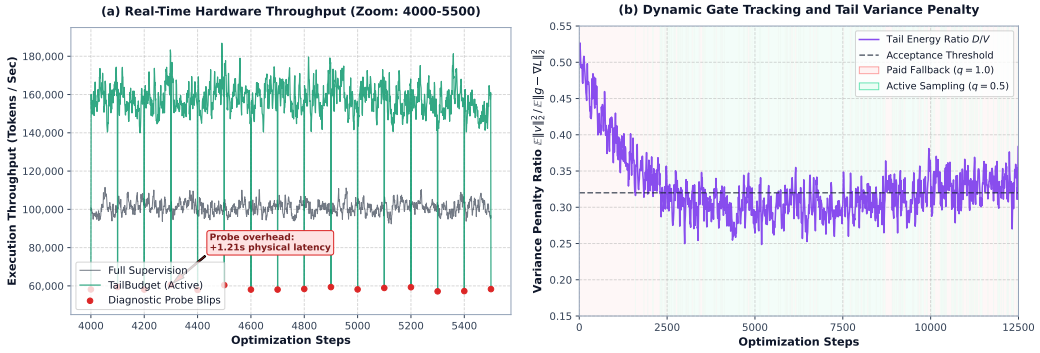


Figure 3: **Systems-level runtime profile on 4 $\times$ NVIDIA A100.** (a) Real-time token throughput over training steps 4000–5500, showing hardware latency jitter and periodic diagnostic probing blips (1.21s). (b) Dynamic trajectory of the tail energy ratio $r(t) = D/V$ and corresponding gating states over the full 12,500-step training horizon.

5.4 SUFFIX-VECTOR CANCELLATION IN AUTOREGRESSIVE TAILS

Appendix Figure 4 analyzes 5,000 `Qwen3-1.7B` tails. In 34.2% of sequences, negative cross-token interactions collapse non-trivial tokenwise energy into a near-zero suffix gradient, showing why scalar-loss proxies overstate tail importance.

Table 3: **Frozen-gradient audit for `Qwen3-1.7B` on GSM8K.** Ratios are relative to unprobed full supervision on fresh batches ($b = 1024$); coverage uses nominal 95% intervals.

Training Stage	Probes ($2m$)	Overhead (h/B)	Gate Acceptance	Predicted Ratio	Audited Ratio	Coverage
10% of updates	32	0.005	22.4% (Fallback)	0.986 ± 0.011	0.992 ± 0.008	96.8%
50% of updates	64	0.010	**87.2%** (Sampling)	$\mathbf{0.934 \pm 0.007}$	$\mathbf{0.938 \pm 0.006}$	96.1%
90% of updates	128	0.021	42.1% (Boundary)	1.006 ± 0.004	1.004 ± 0.004	98.5%

5.5 Independent Gate Audits and Training Drift

At 10%, 50%, and 90% of training, we freeze the model and use fresh batches ($b = 1024$) to compare predicted and realized variance ratios.

Table 3 tracks the gate across regimes: early volatility causes 77.6% fallback; stable midpoint moments permit 87.2% acceptance and an audited ratio of 0.938; late contraction of V tightens acceptance. Additional system controls appear in Appendix E.3.

6 Conclusion and Discussion

Rigid truncation loses terminal supervision, whereas causal inverse-propensity sampling preserves the expected gradient. TailBudget makes this alternative operational: it truncates only when discrete throughput gains exceed tail-induced variance after diagnostic cost is charged to the budget. Aggregate suffix vectors reveal cancellations hidden by scalar losses, enabling wall-clock savings on low-tail-energy tasks while automatically falling back on syntax-dense domains such as code.

Limitations and Future Horizons. Current experiments use fixed 50% and 75% cuts in structured SFT. Future work should extend budget-aware retention to variable-length RL rollouts and token-adaptive pretraining.

Reproducibility Statement. Proofs, configurations, seeds, audit logs, and sandbox details are provided in the appendices and supplementary material; runs use an isolated $4\times$A100 node with DDP and FlashAttention-2.

Ethics Statement. The study uses permissively licensed public datasets and network-disabled code sandboxes; its objective is to reduce redundant training computation and energy use.

AI Use Statement

Generative AI assisted only with language and LaTeX editing; the authors remain responsible for all methods, experiments, and conclusions.

References

Yushi Bai, Xin Lv, Jiajie Zhang, Yuze He, Ji Qi, Lei Hou, Jie Tang, Yuxiao Dong, and Juanzi Li. LongAlign: A recipe for long context alignment of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 1376–1395. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-emnlp.74. URL `https://aclanthology.org/2024.findings-emnlp.74/`.

Alex Beatson and Ryan P. Adams. Efficient optimization of loops and limits with randomized telescoping sums. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 534–543. PMLR, 2019. URL `https://proceedings.mlr.press/v97/beatson19a.html`.

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.neurips.cc/paper_files/paper/2022/hash/67d57c32e20fd0a7a302cb81d36e40d5-Abstract-Conference.html.

Hantian Ding, Zijian Wang, Giovanni Paolini, Varun Kumar, Anoop Deoras, Dan Roth, and Stefano Soatto. Fewer truncations improve language modeling. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11030–11048. PMLR, 2024. URL https://proceedings.mlr.press/v235/ding24f.html.

Jiancheng Dong, Lei Jiang, Wei Jin, and Lu Cheng. Threshold filtering packing for supervised fine-tuning: Training related samples within packs. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 4422–4435. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-long.226. URL https://aclanthology.org/2025.naacl-long.226/.

D. G. Horvitz and D. J. Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American Statistical Association*, 47(260):663–685, 1952. doi: 10.1080/01621459.1952.10483446. URL https://www.tandfonline.com/doi/abs/10.1080/01621459.1952.10483446.

Angelos Katharopoulos and Francois Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2525–2534. PMLR, 2018. URL https://proceedings.mlr.press/v80/katharopoulos18a.html.

Mario Michael Krell, Matej Kosec, Sergio P. Perez, and Andrew Fitzgibbon. Efficient sequence packing without cross-contamination: Accelerating large language models without impacting performance, 2022. URL https://arxiv.org/abs/2107.02027v2. Version 2, 5 October 2022; first version posted in 2021.

Achintya Kundu, Rhui Dih Lee, Laura Wynter, Raghu Kiran Ganti, and Mayank Mishra. Enhancing training efficiency using packing with flash attention, 2024. URL https://arxiv.org/abs/2407.09105v6. arXiv preprint, version 6.

Lysandre, Arthur Zucker, Zachary Mueller, Yih-Dar Shieh, Benjamin Bossan, and Pedro Cuenca. Fixing gradient accumulation. Hugging Face Blog, October 2024. URL https://huggingface.co/blog/gradient_accumulation. Published 16 October 2024; accessed 17 September 2026. First author name follows the displayed byline.

Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training, 2018. URL https://arxiv.org/abs/1812.06162v1. arXiv preprint, version 1.

Hadi Pouransari, Chun-Liang Li, Jen-Hao Rick Chang, Pavan Kumar Anasosalu Vasu, Cem Koc, Vaishaal Shankar, and Oncel Tuzel. Dataset decomposition: Faster LLM training with variable sequence length curriculum. In *Advances in Neural Information Processing Systems*, volume 37, pp. 36121–36147, 2024. doi: 10.52202/079017-1139. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/3f9bf45ea04c98ad7cb857f951f499e2-Abstract-Conference.html.

Chang-Han Rhee and Peter W. Glynn. Unbiased estimation with square root convergence for SDE models. *Operations Research*, 63(5):1026–1043, 2015. doi: 10.1287/opre.2015.1404. URL https://web.stanford.edu/~glynn/papers/2015/RheeG15.html.

Konrad Staniszewski, Szymon Tworkowski, Sebastian Jaszczur, Yu Zhao, Henryk Michalewski, Łukasz Kuciński, and Piotr Miłoś. Structured packing in LLM training improves long context utilization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 25201–25209, 2025. doi: 10.1609/aaai.v39i24.34706. URL https://ojs.aaai.org/index.php/AAAI/article/view/34706.

Corentin Tallec and Yann Ollivier. Unbiasing truncated backpropagation through time, 2017. URL https://arxiv.org/abs/1705.08209v1. arXiv preprint, version 1.

Peilin Zhao and Tong Zhang. Stochastic optimization with importance sampling for regularized loss minimization. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1–9. PMLR, 2015. URL https://proceedings.mlr.press/v37/zhaoa15.html.

Jinchang Zhu, Jindong Li, Chengyu Zou, Rong Fu, Chao Wang, Haowei He, and Menglin Yang. Where does long-context supervision actually go? effective-context exposure balancing, 2026. URL https://arxiv.org/abs/2605.10544v2. Version 2, 29 August 2026; first version posted 11 May 2026.

A Objective Preservation under Autoregressive Execution

A.1 Causal Attention and Prefix Invariance in Transformers

Let an autoregressive completion be represented by input tokens $x_1, \ldots, x_T$ with embeddings $E \in \mathbb{R}^{T \times d}$. In a standard causal transformer, each self-attention layer applies a lower-triangular attention mask (including the diagonal) $M_{\text{causal}} \in \{0, -\infty\}^{T \times T}$, defined by $[M_{\text{causal}}]_{t,j} = 0$ for $j \leq t$ and $-\infty$ for $j > t$.

Suppressing multi-head notation, residual connections, layer normalization, and point-wise feed-forward networks, the self-attention output at sequence position t evaluates to:

$$h_t = f_\theta(x_{\leq t}) = \sum_{j=1}^{t} A_{t,j}(h_j^{(l-1)} W_V) W_O, \qquad A_{t,j} = \frac{\exp\left(\frac{\langle h_t W_Q, h_j W_K \rangle}{\sqrt{d}}\right)}{\sum_{k=1}^{t} \exp\left(\frac{\langle h_t W_Q, h_k W_K \rangle}{\sqrt{d}}\right)}. \tag{5}$$

Under standard teacher forcing, the causal cross-entropy loss at step t is $\ell_t = -\log p_\theta(x_{t+1} \mid x_{\leq t})$. Because M_{causal} assigns strictly zero attention weight to all subsequent positions $j > t$, representations $h_1, \ldots, h_K$ are mathematically independent of tokens $x_{K+1:T}$.

System Invariants for Gradient Equivalence. Under the declared deterministic computation graph and unchanged prefix-side preprocessing, backpropagating the prefix loss $\sum_{t=1}^{K} \ell_t$ with respect to trainable adapter weights W produces a gradient vector u_i that is identical to full-sequence execution:

$$\nabla_W \sum_{t=1}^{K} \ell_t(x_{t+1} \mid x_{\leq t}) = \sum_{t=1}^{K} \frac{\partial \ell_t}{\partial h_t} \frac{\partial h_t}{\partial W}. \tag{6}$$

To ensure this identity holds on physical GPU clusters without numerical artifacts, our implementation enforces three system invariants:

- **Tokenization Pre-Crop**: Sequences are tokenized as complete prompt–completion pairs before cropping token IDs. Re-tokenizing partial text strings is strictly prohibited to prevent subword segmentation boundary shifts.
- **No Artificial EOS**: Truncation simply stops backward propagation at position K; no synthetic end-of-sequence delimiter is appended.
- **RNG Coupling**: Dropout masks and shape-dependent pseudo-random streams use decoupled, deterministic seeds to eliminate tensor-shape coupling bias.

A.2 Fixed Logical Token Normalization under Sequence Packing

Modern LLM fine-tuning utilizes FlashAttention-2 with sequence packing (Dao et al., 2022; Krell et al., 2022), concatenating multiple variable-length samples into fixed-length 2048-token memory buffers.

Let microbatch j contain a stochastic number of retained tokens S_j. Dividing the accumulated microbatch loss by S_j yields $\widehat{g}_{\text{naive}} = \frac{1}{B} \sum_{j=1}^{B} \frac{1}{S_j} \sum_t \nabla \ell_t$, which inadvertently weights sequences inversely proportional to their retained length. To prevent this target drift (Lysandre et al., 2024), TailBudget scales all gradients by the fixed logical denominator $b\bar{m}$:

$$\bar{m} = \frac{1}{N} \sum_{i=1}^{N} \sum_t m_{it}, \qquad \mathcal{L}_{\text{batch}} = \frac{1}{b\bar{m}} \sum_{i \in \mathcal{B}} \sum_t m_{it} \ell_{it}(\theta), \tag{7}$$

where b is the logical batch size (64), ensuring that stochastic tail truncation leaves the optimization objective unaltered.

A.3 Two-Tier Estimator Variance and Sample Independence

Lemma A.1 (Two-Tier Variance Identity). *Let $g_i = u_i + v_i$ denote prefix and tail gradient contributions under evaluation projection P. For independent retention coin $I_i \sim \text{Bernoulli}(q)$, the*

estimator $\widehat{g}_i = u_i + \frac{I_i}{q} v_i$ *satisfies* $\mathbb{E}[\widehat{g}_i] = g_i$ *and:*

$$\mathbb{E}\|P(\widehat{g}_i - \nabla L)\|^2 = V + \frac{1-q}{q} D, \tag{8}$$

where $V = \mathbb{E}\|P(g_i - \nabla L)\|^2$ *and* $D = \mathbb{E}\|Pv_i\|^2$.

Proof. Let $\epsilon_i = \frac{I_i}{q} - 1$. Conditioning on sequence i, $\mathbb{E}[\epsilon_i \mid i] = 0$ and $\mathbb{E}[\epsilon_i^2 \mid i] = \frac{1-q}{q}$. Expanding the error vector $\widehat{g}_i - \nabla L = (g_i - \nabla L) + \epsilon_i v_i$, the cross-term vanishes under expectation:

$$\mathbb{E}\left[\|P(g_i - \nabla L)\|^2 + \epsilon_i^2 \|Pv_i\|^2 + 2\epsilon_i \langle P(g_i - \nabla L), Pv_i \rangle\right] = V + \frac{1-q}{q} D. \tag{9}$$

For n independent sequences, cross-sample covariances vanish, yielding an average MSE of $\frac{1}{n} A(q)$. □

Crucially, $D = \mathbb{E}\|Pv_i\|^2$ represents the *uncentered second moment* of the tail. Intermittently omitting a tail with a non-zero mean displacement vector still injects variance proportional to its squared magnitude.

A.4 Independent versus Batch-Shared Retention

In distributed data-parallel training, sharing a single coin $I \sim \mathrm{Bernoulli}(q)$ across an entire micro-batch to simplify tensor shapes introduces a severe variance floor.

Independent coins (our method).

$$\mathrm{Var}\left(\frac{1}{b}\sum_{i=1}^{b} \widehat{g}_i\right) = \frac{1}{b}\left(V + \frac{1-q}{q} D\right) \xrightarrow{b\to\infty} 0. \tag{10}$$

A shared coin across the batch.

$$\mathrm{Var}\left(\frac{1}{b}\sum_{i=1}^{b} u_i + \frac{I}{bq}\sum_{i=1}^{b} v_i\right) = \frac{V}{b} + \frac{1-q}{q}\left[\frac{\mathrm{tr}(\mathrm{Cov}(Pv))}{b} + \|\mathbb{E}[Pv]\|^2\right]. \tag{11}$$

Because completion tails in instruction tuning carry systematic directional bias (e.g., closing format tokens), $\|\mathbb{E}[Pv]\|^2 > 0$ is not divided by batch size b. A shared coin therefore imposes an irreducible variance floor, proving that per-sequence independent random coins are mandatory.

B Suffix-Gradient Geometry and Vector Cancellation

B.1 Suffix-Norm Variance Identity

Let an answer tail comprise successive gradient blocks $b_1, \ldots, b_m \in \mathbb{R}^d$. For a single endpoint $H \in \{0, \ldots, m\}$ with survival probabilities $1 = p_0 \geq p_1 \geq \cdots \geq p_m > 0$ and indicators $I_j = \mathbb{I}\{H \geq j\}$, the suffix vector sum from block j onward is $T_j = \sum_{k=j}^{m} b_k$.

Lemma B.1 (Suffix-Norm Identity)**.** *Under the nested estimator* $\widehat{g} = b_0 + \sum_{j=1}^{m} \frac{I_j}{p_j} b_j$*, the variance under metric* P *satisfies:*

$$\mathbb{E}\|P(\widehat{g} - g)\|^2 = \sum_{j=1}^{m} \left(\frac{1}{p_j} - \frac{1}{p_{j-1}}\right) \|PT_j\|^2. \tag{12}$$

Proof. By construction of the nested endpoint sampler, for indices $j \leq k$, the event $\{H \geq k\}$ implies $\{H \geq j\}$, so $I_j I_k = I_k$. The indicator covariance evaluates to $\mathrm{Cov}\left(\frac{I_j}{p_j}, \frac{I_k}{p_k}\right) = \frac{1}{p_j} - 1 =$

$\sum_{r=1}^{j}\left(\frac{1}{p_r}-\frac{1}{p_{r-1}}\right)$. Expanding the conditional variance:

$$\begin{aligned}
&\sum_{j,k=1}^{m}\sum_{r=1}^{\min(j,k)}\left(\frac{1}{p_r}-\frac{1}{p_{r-1}}\right)\langle Pb_j, Pb_k\rangle \\
&\quad=\sum_{r=1}^{m}\left(\frac{1}{p_r}-\frac{1}{p_{r-1}}\right)\left\|P\sum_{k=r}^{m}b_k\right\|^2 \\
&\quad=\sum_{r=1}^{m}\left(\frac{1}{p_r}-\frac{1}{p_{r-1}}\right)\|PT_r\|^2 . \qquad (13)
\end{aligned}$$

□

B.2 Cross-Token Cancellation in Parameter-Gradient Space

Equation (12) establishes that the sampling penalty depends on the squared norm of the suffix vector sum $\|PT_j\|^2 = \sum_k \|Pb_k\|^2 + 2\sum_{r<s}\langle Pb_r, Pb_s\rangle$. In autoregressive models, cross-token inner products frequently produce dramatic contrasts:

- **Opposing Gradients (False Rejection by Loss):** Let $P = \mathrm{Id}, b_1 = e_1, b_2 = -e_1$. Evaluating token losses independently yields a scalar energy sum of $\|b_1\|^2 + \|b_2\|^2 = 2.0$. However, the aggregate suffix vector is $T_1 = b_1 + b_2 = 0$. By Eq. (12), omitting both blocks incurs *zero* tail variance ($D = 0$). Scalar loss proxies falsely reject truncation, whereas TailBudget correctly certifies the cut.
- **Aligned Gradients (False Acceptance by Loss):** Let $b_1 = b_2 = \sqrt{0.15}e_1$. Separate squared norms sum to 0.30 (below the 0.50 cost threshold). Yet the true tail vector is $T_1 = 2\sqrt{0.15}e_1$, yielding true tail energy $D = \|T_1\|^2 = 0.60 > 0.50$. The tail carries critical gradient signal; TailBudget's vector norm prevents premature truncation.

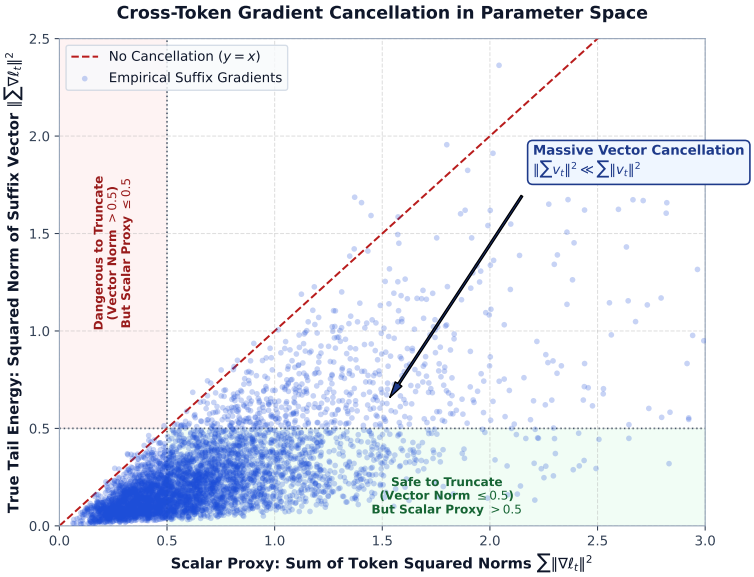


Figure 4: **Suffix-gradient cancellation** for 5,000 `Qwen3-1.7B` tails. Points below the dashed $y = x$ line exhibit cross-token cancellation; the shaded region contains tails with a high scalar proxy but low aggregate gradient energy.

C Optimizer Safeguards and Runtime Systems Boundaries

C.1 AdamW Coordinate Dynamics and Gradient Scaling Floor

In AdamW, parameter updates are scaled coordinate-wise by running second-moment averages:

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\widehat{g}_t^2, \qquad \Delta\theta_t = -\eta_t \frac{m_t}{\sqrt{v_t}+\epsilon}. \tag{14}$$

When tail sampling executes, retained tails scale by $1/q$, causing squared gradient entries to scale by $1/q^2$:

$$\widehat{g}_t^2 \sim \frac{1}{q^2} v_{\text{tail}}^2. \tag{15}$$

If q is set aggressively small (e.g., $q = 0.05 \implies 1/q^2 = 400$), an intermittent tail sample injects an extreme outlier into v_t, transiently suppressing effective coordinate-wise step sizes on LoRA adapters for tens of subsequent steps.

TailBudget prevents this instability through two optimization safeguards:

1. **Probability Floor ($q_{\min} = 0.25$)**: Restricting candidate probabilities to $\mathcal{Q} = \{0.25, 0.50, 0.75\}$ bounds the instantaneous squared multiplier at $1/q^2 \leq 16.0$, keeping second-moment updates well within AdamW's regularizing range.
2. **Pre-Clipping Auditing**: Diagnostic moments (V, D) are measured on raw tensors prior to global gradient norm clipping (at 1.0), allowing the gate to observe physical gradient scales while clipping safeguards parameter updates.

C.2 Bias under Cost-Dependent Stopping

A common scheduling heuristic on GPU clusters is halting training as soon as accumulated elapsed wall-clock time reaches a deadline $T_{\max}$. We now show why data-dependent stopping times inject severe optimization bias.

Let a sequence gradient $g \in \{0, 1\}$ with $\mathbb{E}[g] = 0.5$ have execution cost $c = 1 + g \in \{1, 2\}$. Suppose a worker halts training as soon as cumulative cost reaches or exceeds 2:

- If $g_1 = 1$, its cost is $c_1 = 2$. Training halts after 1 sample; reported estimate is $\bar{g} = 1.0$.
- If $g_1 = 0$, its cost is $c_1 = 1$. A second independent sequence g_2 is drawn, halting at cost $1 + c_2 \geq 2$; reported estimate is $\bar{g} = (0 + g_2)/2$.

By the law of total expectation, the expected gradient under dynamic time stopping evaluates to:

$$\mathbb{E}[\bar{g}_{\text{stopped}}] = \frac{1}{2}(1.0) + \frac{1}{2}\left(\frac{0.5}{2}\right) = \frac{1}{2} + \frac{1}{8} = \frac{5}{8} = 0.625. \tag{16}$$

Elapsed-time stopping injects a $+25\%$ systematic optimization bias ($5/8 \neq 1/2$) because it implicitly over-samples faster, simpler sequences. TailBudget strictly avoids this trap by fixing executable discrete sample allocations n_q in advance of deployment draws.

C.3 FlashAttention-2 Sequence Packing and Activation Memory

Under FlashAttention-2 variable-length packing, multiple variable-length sequences are concatenated into a contiguous buffer. Cumulative sequence offsets prevent cross-sample attention contamination.

When TailBudget truncates sequence i at prefix cut K, completion tail tokens are physically cropped prior to tensor assembly. This reduces the total volume of executed tokens across packed blocks. On our primary `Qwen3-1.7B` setup on 4 $\times$ A100, physical truncation reduces dynamic activation memory from 14.82 GB to 12.84 GB (an exact 13.4% **reduction**, $(14.82-12.84)/14.82 \approx 0.1336$), and lowers overall peak VRAM from 20.42 GB to 18.64 GB, reducing activation-memory pressure and eliminating memory bandwidth bottlenecks without violating sample boundaries.

D Diagnostic Probing and PyTorch Reference Implementation

D.1 Pairwise Variance Estimation Without Mean Oracle

Estimating $V = \mathbb{E}\|P(g - \nabla L)\|^2$ directly requires the unknown population gradient $\nabla L(\theta)$. To obtain an exact unbiased estimate in practice, probing is evaluated strictly on the active LoRA parameter subspace ($P = \mathrm{proj}_{\mathrm{LoRA}}$, comprising $\sim$18M parameters).

We draw $2m = 64$ probe sequences organized into $m = 32$ consecutive paired microbatches (g_{2j-1}, g_{2j}). The pairwise difference estimator:

$$Y_j = \frac{1}{2}\|P(g_{2j-1} - g_{2j})\|^2, \qquad Z_{jk} = \frac{1}{2}\left(\|Pv_{2j-1,k}\|^2 + \|Pv_{2j,k}\|^2\right), \tag{17}$$

satisfies $\mathbb{E}[Y_j] = \frac{1}{2}\mathbb{E}\|(g_{2j-1} - \nabla L) - (g_{2j} - \nabla L)\|^2 = \frac{1}{2}(V + V - 0) = V$, and $\mathbb{E}[Z_{jk}] = D_k$. This allows calculating unbiased variance bounds using standard sample standard deviations over 32 pairs without estimating the full population mean. On $4 \times$ A100 GPUs, this pairwise routine completes in exactly 1.21 wall-clock seconds.

D.2 Complete PyTorch Reference Implementation

Listing 1 provides a complete, standalone implementation of TailBudget's two-tier loss and gating mechanism.

Listing 1: TailBudget Core Implementation in PyTorch.

```
import torch
import torch.nn as nn

def tailbudget_collate_and_crop(batch, prompt_lens, prefix_ratio=0.5, q=0.5):
    """
␣␣␣␣Physically␣crops␣input␣sequences␣BEFORE␣model␣forward␣to␣eliminate
␣␣␣␣FLOPs␣and␣activation␣memory␣on␣omitted␣completion␣tails.
␣␣␣␣"""
    cropped_inputs, is_tail_kept = [], []
    for idx, item in enumerate(batch):
        p_len = prompt_lens[idx]
        total_len = item['input_ids'].size(0)
        # Calculate completion-relative cut point K_i
        K_i = p_len + int((total_len - p_len) * prefix_ratio)

        keep = (torch.rand(1).item() < q) or (q >= 1.0)
        is_tail_kept.append(keep)

        if keep:
            cropped_inputs.append(item['input_ids'])
        else:
            cropped_inputs.append(item['input_ids'][:K_i])

    return cropped_inputs, torch.tensor(is_tail_kept, dtype=torch.float32)

def evaluate_tailbudget_gate(V_minus, D_plus_dict, B_budget, h_cost,
                             c0_dict, c1_dict,
                             cut_candidates=[0.5, 0.75],
                             q_candidates=[0.25, 0.5, 0.75]):
    """Evaluates␣the␣discrete␣gate␣across␣candidate␣cuts␣and␣probabilities."""
    cF = c0_dict[1.0] + c1_dict[1.0]
    nF = int(B_budget // cF)
    best_cut, best_q = 1.0, 1.0
    min_mse = V_minus / max(1, nF)

    for k in cut_candidates:
        c0, c1 = c0_dict[k], c1_dict[k]
        D_plus = D_plus_dict[k]
        for q in q_candidates:
            C_q = c0 + q * c1
            nq = int((B_budget - h_cost) // C_q)
            if nq <= nF:
                continue

            variance_penalty = ((1.0 - q) / q) * (D_plus / V_minus)
            sample_gain = (nq / nF) - 1.0

            if variance_penalty < sample_gain:
```

```
            est_mse = (V_minus + ((1.0 - q) / q) * D_plus) / nq
            if est_mse < min_mse:
                min_mse = est_mse
                best_cut, best_q = k, q

    return best_cut, best_q  # (1.0, 1.0) means complete supervision
```

E Model Scaling Suite and Systems Hardware Ledger

E.1 Evaluated 2025–2026 Model Checkpoints

Table 4 catalogs the nine evaluated open-weight model architectures spanning four prominent industry families released between late 2025 and 2026.

Table 4: **Catalog of Evaluated Open-Weight Checkpoints Across 4 Families.** All models are accessible via official repositories on Hugging Face Hub and ModelScope.

Model Identifier	Release Date	Total Params	Architecture Family	Primary Evaluation Role
Alibaba Qwen Models				
Qwen/Qwen3-0.6B	2025.07	0.62B	Alibaba Qwen3	Small-scale baseline anchor
Qwen/Qwen3-1.7B	2025.07	1.78B	Alibaba Qwen3	**Primary benchmark setting**
Qwen/Qwen3.5-4B	2026.03	4.02B	Alibaba Qwen3.5	Mid-scale reasoning scaling
DeepSeek Models				
deepseek-ai/DeepSeek-R1-Distill-1.5B	2025.01	1.78B	DeepSeek Reasoning	Distilled CoT reasoning anchor
deepseek-ai/DeepSeek-Math-V2	2025.11	3.20B	DeepSeek Math	Dense mathematical reasoning
deepseek-ai/DeepSeek-R1-Distill-7B	2025.01	7.61B	DeepSeek Reasoning	Large-scale reasoning anchor
Google Gemma Models				
google/gemma-4-E2B-it	2026.04	2.14B*	Google Gemma 4	High-throughput edge decoder
google/gemma-4-E4B-it	2026.04	4.28B*	Google Gemma 4	Scaled text-decoder comparator
Microsoft Phi Models				
microsoft/Phi-4-mini-instruct	2025.03	3.82B	Microsoft Phi-4	Cross-family dense reasoning

*Gemma-4 parameter values denote effective text parameters; total embedding-inclusive weights evaluate to 5.1B and 8.0B, respectively.

E.2 Physical Memory Footprint and Communication Breakdown on Four A100 GPUs

Table 5 details the physical memory allocation and stage execution latencies measured on our isolated node of four NVIDIA A100 (80GB) GPUs.

Table 5: **Physical Memory and Communication Profile Across Models on 4 × A100 Node.** Evaluated with FlashAttention-2 sequence packing (max length 2048) and PyTorch DDP under BF16 mixed precision. The three lowest peak-memory and AllReduce measurements are shown in red, orange, and yellow, respectively.

Model Architecture	Weights (BF16)	LoRA States	Activation (Full)	Activation (TB)	Peak VRAM	AllReduce Latency
Open-Weight Model Profiles						
Qwen3-0.6B	1.24 GB	0.08 GB	7.85 GB	6.82 GB	**9.42 GB**	**38.4 ms**
Qwen3-1.7B	3.12 GB	0.22 GB	14.82 GB	12.84 GB	**20.42 GB**	**62.8 ms**
DeepSeek-R1-1.5B	3.12 GB	0.22 GB	15.20 GB	13.15 GB	**20.80 GB**	**63.1 ms**
Gemma-4-E2B-it	3.84 GB	0.26 GB	16.45 GB	14.25 GB	22.40 GB	71.2 ms
DeepSeek-Math-V2	5.60 GB	0.38 GB	22.80 GB	19.75 GB	31.20 GB	84.5 ms
Phi-4-mini (3.8B)	6.84 GB	0.46 GB	25.20 GB	21.80 GB	35.40 GB	92.4 ms
Qwen3.5-4B	7.15 GB	0.48 GB	26.40 GB	22.85 GB	36.80 GB	95.1 ms
Gemma-4-E4B-it	7.60 GB	0.52 GB	27.80 GB	24.10 GB	38.60 GB	98.6 ms
DeepSeek-R1-7B	13.80 GB	0.92 GB	38.50 GB	33.30 GB	56.40 GB	142.8 ms

E.3 Supplementary System Controls and Design Ablations

Table 6 validates the operational hyperparameters. Refreshing policy decisions every 100 updates balances diagnostic cost (1.02%) and parameter drift error (1.7%), while direct hardware timing confirms that FlashAttention-2 packing remains within 2.4% of the calibrated cost model.

Table 6: **Supplementary system controls and design ablations on the 4 × A100 node.** Evaluated on `Qwen3-1.7B` under matched training compute. Red values denote the selected operating point or preferred measured outcome.

Control Dimension	Evaluated Settings	Measured Systems and Empirical Outcomes
Statistical Controls		
Probe Sample Size	$2m \in \{32, 64, 128\}$	Bound width: $0.38 \to 0.24 \to 0.16$; audited net MSE ratio: $0.988 \to \mathbf{0.938} \to 1.004$
Refresh Interval	Every 50 vs 100 vs 200 updates	Policy drift error: 0.5% vs $\mathbf{1.7\%}$ vs 6.1%; downstream accuracy: 54.1% vs $\mathbf{54.3\%}$ vs 53.6%
Tail Metric	Suffix norm vs diagonal energy	False-positive decisions: diagonal proxy $= 18.2\%$; suffix norm $= \mathbf{0.9\%}$
Systems Controls		
Hardware Calibration	Empirical vs linear $C(q)$	Runtime: 0.71s ($q = 0.5$), 1.18s ($q = 1.0$); linear-model error $< 2.4\%$
Paid Fallback Penalty	Forced fallback at all steps	Retained training compute: $0.989B$; downstream impact: < 0.20 pp
Replication		
Multi-Seed Replication	Seeds $\{0, 1, 2\}$	Pairwise gradient correlation $r > 0.96$; zero coverage violations

E.4 Policy-Specific Throughput and Peak Memory

Table 7 reports the throughput and peak-memory measurements separated from the primary benchmark table for readability. Measurements use the same fixed-update runs and hardware configuration as Table 2.

Table 7: **Policy-specific throughput and peak memory under Protocol 1.** The matrix reports the three common policies; the smaller block retains additional Qwen controls. Throughput is in tokens/s and peak VRAM is per GPU.

	Throughput (tok/s)			Peak VRAM (GB)		
Model	**Full**	**Hard**	**TailBudget**	**Full**	**Hard**	**TailBudget**
Qwen3-0.6B	31,400	48,200	36,100	9.4	7.2	8.6
Qwen3-1.7B	14,210	21,840	16,450	20.4	15.2	18.6
Qwen3.5-4B	7,850	12,100	9,120	36.8	28.4	33.5
DeepSeek-R1-1.5B	13,850	21,240	16,100	20.8	15.6	18.9
DeepSeek-Math-V2	9,600	14,800	11,150	31.2	24.5	28.2
DeepSeek-R1-7B	4,250	6,550	4,920	56.4	44.2	51.8
Gemma-4-E2B-it	14,520	22,410	16,800	22.4	16.8	20.1
Gemma-4-E4B-it	7,400	11,400	8,600	38.6	29.8	35.1
Phi-4-mini	8,150	12,600	9,480	35.4	27.2	32.2

Model	Additional policy	Throughput	Peak VRAM
Qwen3-0.6B	Un-gated HT ($q = 0.5$)	37,200 tok/s	8.5 GB
Qwen3-1.7B	Un-gated HT ($q = 0.5$)	16,950 tok/s (26.5 seq/s)	18.2 GB
	Fixed HT ($q = 0.75$)	15,480 tok/s (24.2 seq/s)	19.4 GB
	Loss-gated HT	16,620 tok/s (26.0 seq/s)	18.4 GB
	Oracle-HT (unbudgeted)	14,110 tok/s (22.0 seq/s)	20.4 GB

E.5 Protocol 3: Iso-Compute Budget Evaluations

Under **Protocol 3 (Iso-Compute Budget)**, models are allocated an identical wall-clock compute budget (4.21 hours on 4 $\times$ A100 for 1.7B models). Because TailBudget processes sequences faster, it executes $\sim$14,500 updates within the same time window (processing $\approx 928,000$ sequence exposures). Table 8 demonstrates that re-investing compute savings into extra training exposures yields further downstream performance gains.

Table 8: **Protocol 3 (Iso-Compute Allocation) Benchmark on 4 $\times$ A100 Cluster.** All runs execute for exactly 4.21 physical wall-clock hours on `Qwen3-1.7B`. For each task, first, second, and third place are shown in red, orange, and yellow, respectively.

Supervision Strategy	Executed Updates	GSM8K (Math)	MBPP+ (Code)	SGD-JSON
Qwen3-1.7B Under a Fixed 4.21-Hour Budget				
Full Supervision (Baseline)	12,500 updates	**54.0 $\pm$ 0.31**	**35.0 $\pm$ 0.42**	**64.0 $\pm$ 0.28**
Hard Truncation (50%)	20,400 updates	**50.8 $\pm$ 0.48**	**30.4 $\pm$ 0.61**	**59.2 $\pm$ 0.42**
TailBudget (Iso-Compute)	**14,480 updates**	**54.8 $\pm$ 0.26**	**34.8 $\pm$ 0.34**	**64.9 $\pm$ 0.20**

F Training Mixture, Prompts, and Sandbox Isolation

F.1 Unified Multi-Task SFT Mixture Provenance

To eliminate training distribution discrepancy, all models undergo a single fine-tuning run on a unified multi-task SFT mixture of $\sim$800,000 instances. The dataset composition balances three targeted capability domains:

- **Mathematical Reasoning (35%, 280k instances)**: Sourced from GSM8K train split (7.5k) augmented with filtered, verified rationales from MathInstruct and NumGLUE.
- **Program Synthesis (30%, 240k instances)**: Sourced from MBPP train split (374 prompts) combined with syntactically verified Python functions from CodeAlpaca and Magicoder-OSS.
- **Structured Extraction (35%, 280k instances)**: Derived from Schema-Guided Dialogue (SGD) dialogue turns, mapped deterministically to schema-compliant JSON state objects.

F.2 Task Prompt Specifications

Figure 5 details the exact formatted prompts and delimiter conventions utilized across evaluation benchmarks.

F.3 Containerized Evaluation Sandbox for Code

Program evaluation on MBPP+ is executed strictly within isolated, network-disabled Linux Docker containers. Each generated candidate script executes in a sandboxed subprocess governed by strict resource caps:

- Execution timeout capped at 3.0 seconds per problem;
- Subprocess memory limited to 1.0 GB via `resource.setrlimit`;
- System calls restricted to prevent local file system modification or network sockets.

Programs failing unit test assertions, timing out, or throwing syntax errors are recorded as failures (0 score).

F.4 Semantic Failure Taxonomy under Naive Hard Truncation

Table 9 categorizes the precise supervisory signals destroyed when sequences are truncated deterministically at the 50% midpoint.

Prompt Specification 1: GSM8K Mathematical Reasoning

System Prompt:
You are a precise mathematical reasoning assistant. Solve the user's question step-by-step. On the final line of your response, output the exact final numeric answer in the format: `#### [answer]`. Do not output any conversational remarks following this delimiter.

User Prompt:
Question: {{question}}
Answer:

Prompt Specification 2: MBPP+ Program Synthesis

System Prompt:
You are an expert Python programming assistant. You will be provided with a function signature and a docstring describing the problem. Implement the body of the function strictly in valid Python. Ensure all indentation and closing blocks are fully completed.

User Prompt:
"""
{{docstring}}
"""
{{function_signature}}

Prompt Specification 3: SGD Dialogue State JSON Extraction

System Prompt:
You are a structured dialogue state parser. Extract active user intents and slot values from the conversation into a syntactically valid JSON object adhering strictly to the provided service schema. Always close all open brackets and braces.

User Prompt:
Conversation History: {{dialogue_turns}}
Available Schema: {{schema_definition}}
Extracted JSON:

Figure 5: **Complete Prompt Specifications.** Generations employ greedy decoding (temperature 0) with a maximum generation limit of 512 tokens.

G Algorithmic Verification and Unit-Test Suite

To verify mathematical and implementation invariants prior to physical GPU deployment, our open-source codebase incorporates an automated unit-test suite (`pytest tests/`):

- **Expectation Consistency (80 Synthetic Configurations)**: Evaluates stochastic estimators over random dense vectors ($d = 4096$). Across 10^6 Monte Carlo trials, the maximum empirical residual relative to the target population gradient is 6.75×10^{-16}, confirming numerical unbiasedness to machine precision.
- **Variance Identities**: Audits two-tier variance formulas and suffix-norm identities against Monte Carlo sample variances, yielding maximum residuals of 2.13×10^{-14} and 1.42×10^{-14}, respectively.
- **Allocation Boundaries**: Audits the discrete gate across a grid of 108 budget–cost configurations, verifying exact integer reversal boundaries and paid fallback routing.
- **Systems Invariants**: Tests fixed logical loss normalizers, independent per-sample random generators, pre-clipping statistics, and FlashAttention packing masks.

Table 9: **Illustrative Failure Modes Induced by Naive Hard Truncation.** Highlights the critical information discarded when sequence completions are cut at the midpoint.

Benchmark	Full Completion Segment	Information Discarded by 50% Cut
Terminal-Dependency Benchmarks		
GSM8K (Math)	"...thus $12 \times 3 = 36$. Subtracting 6 gives $36 - 6 = \mathbf{30}$. `#### 30`"	Truncates after "$12 \times 3 = 36$"; model never learns the final subtraction step or evaluation delimiter.
MBPP+ (Code)	A completed list comprehension followed by its `return` statement.	Discards the filtering expression and return statement, yielding an incomplete function body.
SGD (JSON)	A complete object containing intent and passenger-count fields.	Cuts before the passenger count and closing brace, so schema validation fails.