FAST LAPSUM: EXACT DIFFERENTIABLE TOP-k AT MILLION SCALE

Anonymous authors
Paper under double-blind review

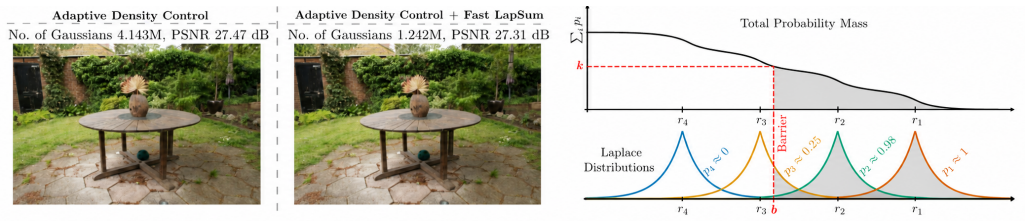

Figure 1: With Fast LapSum, we achieve almost **two times faster** rendering, using **70%** less memory than a baseline in 3D Gaussian Splatting pruning, with nearly no change in visual quality. Our solution scales to **millions of elements**.

ABSTRACT

Selecting the top-k elements is a fundamental operation for inducing sparsity in large-scale models and optimization problems, enabling robust expert activation, token routing or attention pruning. However, hard top-k is non-differentiable, while existing differentiable alternatives become increasingly expensive as the number of coordinates grows. We introduce Fast LapSum, a scalable solver for the LapSum soft top-k formulation that preserves an exact selection mass of k, while supporting end-to-end differentiation. In Fast LapSum, we reduce sorting cost using probabilistic bracketing, which restricts sorting to a narrow band of scores around the threshold using a binomial order-statistic from kernel-noised samples. A certification pass upgrades the probabilistic localization to a verified one at the cost of one additional linear pass, with a full-sort fallback that covers the worst-case scenario. Our certified GPU implementation processes 10^6, 10^7, and 10^8 scores in median times of 0.92, 1.39, and 7.24 ms, respectively, making exact-budget soft top-k practical within million-scale optimization loops. We demonstrate this capability in two applications: megapixel sparse adversarial examples with a small fraction of initial image pixels, where Fast LapSum achieves an order-of-magnitude speedup over state-of-the-art methods, and 3D Gaussian splatting. In the latter, we use Fast LapSum to reduce the number of Gaussians produced by Adaptive Density Control to a substantially smaller number while retaining nearly the same rendering quality and massively reducing computation. These results demonstrate that exact-budget differentiable top-k can be incorporated into practical million-scale optimization pipelines.

1 INTRODUCTION

Sparse computation has become a principal approach to scaling neural networks. Mixture-of-experts models activate only a few experts, long-context models retain a limited set of tokens or memories, retrieval systems score millions of candidates and return a small shortlist, and sparse attention or graph layers select only a small neighbourhood. These systems share a single operation, top-k selection, which converts a dense score vector into a sparse, lower-cost subset.

Yet that same operation is a persistent obstacle to end-to-end learning. Hard top-k yields the desired sparsity but almost no useful gradient through the decision boundary. Soft relaxations restore gradients, but often at the cost of dense matrices, transport solves, sorting-like complexity, or masks

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.

whose total mass is not directly controllable. For sparse neural systems this is the wrong trade-off: a differentiable sparsifier that is itself expensive merely relocates the bottleneck. We argue that differentiable top-k should be treated as a systems primitive, keeping the exact budget of hard top-k, exposing useful gradients, and running cheaply enough to sit inside the training loop. This paper provides such a primitive by making the exact LapSum operator (Struski et al., 2025) GPU-native and million-scale. Given scores $r = (r_1, \dots, r_n) \in \mathbb{R}^n$, budget k, and temperature $\alpha < 0$, LapSum returns probabilities

$$p_i = \sigma\left(\frac{b - r_i}{\alpha}\right), \qquad \sum_{i=1}^{n} p_i = k, \tag{1}$$

where σ is the Laplace CDF and the scalar threshold b enforces the budget exactly. Throughout, *budget-exact* refers to this normalization of the mask, not to the cardinality of its support, which coincides with k only in the zero-temperature limit (Sec. 3).

Previous GPU implementations solve for b by bisection and re-sum all n terms at every step. Fast LapSum, presented in Fig. 1, computes the entire threshold curve. After an initial sort, one stable prefix/suffix scan yields the curve, and the threshold follows from an index lookup and a closed-form root. The backward pass reuses the analytical vector–Jacobian product of Struski et al. (2025) unchanged. This alone gives nearly two orders of magnitude speed-up over the per-query bisection baseline. For million-scale vectors, where sorting itself becomes the bottleneck, we add a binomial order-statistic bracket that localizes the threshold, streams the vector once, and sorts only the uncertain middle region, still recovering the same b. The i.i.d. retry variant has expected linear running time under the assumptions of Prop. 3. The deployed systematic path is evaluated empirically and uses the full-sort solver as an exact $\mathcal{O}(n \log n)$ fallback. The certified implementation stays in the low milliseconds across the tested large-n range and is much more efficient than the reference LapSum.

We evaluate the operator in two settings with millions of coordinates. In 3D Gaussian Splatting, a soft score-learning stage selects an exact hard primitive subset and reduces the deployed representation. In adversarial attack optimization, a hard-forward straight-through estimator changes at most K pixels while Fast LapSum supplies the score gradient. These applications utilize both soft-to-hard pruning and exact-support training. The most closely related concurrent work is DFTopK (Zhu et al., 2025), which also targets linear-time differentiable top-k but relaxes $\sum_i p_i = k$. We retain the exact normalization and extend the scaling study three orders of magnitude beyond its largest tested size (Sec. F.1).

Our contribution concerns the threshold computation and the associated large-scale pipeline:

- **Fast LapSum is an exact-budget differentiable top-k operator working with expected linear time ($\mathcal{O}(n)$).** In this category, it is the only method that preserves the unrelaxed constraint $\sum_i p_i = k$.
- **A probabilistic bracketing pipeline for million-scale vectors.** We develop a probabilistic sample-bracketing mechanism that isolates the exact solver to a narrow uncertainty range. Supported by deterministic certification, it guarantees strict budget exactness. The certified implementation is $457\times$ faster than LapSum at 10^9 scores on an NVIDIA GH200.
- **Fast LapSum is highly practical at the megapixel scale without altering downstream mathematical objectives**. It accelerates high-resolution pixel-L_0 adversarial attacks and enables extreme 3D Gaussian Splatting pruning (even 30% retention), doubling rendering throughput with negligible quality degradation.

2 Related Work

Differentiable Ranking and Top-k. Continuous relaxations of sorting and ranking include NeuralSort (Grover et al., 2019), SoftSort (Prillo & Eisenschlos, 2020), differentiable sorting networks (Petersen et al., 2021), the permutahedron/isotonic operators of Blondel et al. (2020), the entropic optimal-transport formulation of Xie et al. (2020), and convex-analysis sparse top-k (Sander et al., 2023). These return richer objects (soft ranks, soft permutations, projections) at $\mathcal{O}(n^2)$ or iterative cost, whereas our target is narrower and more operational: a budgeted soft mask with the same output shape as hard top-k. We build directly on LapSum (Struski et al., 2025), which unifies ranking,

sorting and top-k through a single Laplace-CDF threshold equation. Concurrently, DFTopK (Zhu et al., 2025) also targets linear-time differentiable top-k for large-scale recommendation by relaxing the normalization constraint: it places the soft threshold at the midpoint $c = \frac{1}{2}(r_{(k)} + r_{(k+1)})$ of the two boundary order statistics, which is fast and linear-time but makes the mask non-budget-exact at finite temperature and lets the threshold gradient reach only $r_{(k)}$ and $r_{(k+1)}$. Prop. 1 precises the relation, and we compare both approaches directly in Sec. 5 and Sec. F.1.

Sampling-Based Selection. Locating an order statistic through a small sample is a classical technique (Blum et al., 1973; Floyd & Rivest, 1975; Alabi et al., 2012). While concurrent work like Prof-K (Dziarmaga et al., 2026) applies this to hard top-k discarding of out-of-bracket elements, our budget-exact soft operator must accumulate mass ($\sum_i p_i = k$) from all scores. Consequently, sampling only serves to localize the region where the exact budget equation is solved (Prop. 3).

Where Differentiable Top-k Is Used. A budgeted differentiable selector is the shared mechanism behind many sparse systems, which nowadays mostly fall back on hard selection or task-specific penalties: mixture-of-experts routing under a fixed capacity (Shazeer et al., 2017; Fedus et al., 2022), adaptive token pruning (Rao et al., 2021), top-k sparse autoencoders (Makhzani & Frey, 2014; Gao et al., 2025), network and movement pruning (Han et al., 2015; Frankle & Carbin, 2019; Sanh et al., 2020), and retrieval, sparse attention or gradient compression at larger candidate scale. In each case, the active set is chosen by a hard top-k that blocks gradients while enforcing a hard budget, precisely the regime an exact-budget, GPU-native, million-scale differentiable top-k is built for. Our experiments evaluate the operator and the two large-scale applications described in Sec. 6.

3 Background: The LapSum Operator

In this section, we restate the LapSum operator described in (Struski et al., 2025) and then define the terminology and Prop. 1. The hard top-k operation relies on a quantile threshold and is harmful during neural network training because moving one score by an arbitrarily small amount can change its mask value discontinuously. LapSum keeps the same logic but introduces a probabilistic test. Each score is perturbed by a Laplace kernel, and p_i is the analytically computed probability that the perturbed copy of r_i falls on the top-k side of the threshold.

Let $t = |\alpha| > 0$ denote the temperature where the convention with $\alpha < 0$ follows (Struski et al., 2025). Given scores $r \in \mathbb{R}^n$, define the *budget curve* $\Psi_r : \mathbb{R} \to (0, n)$ by

$$\Psi_r(b) = \sum_{i=1}^{n} \sigma\left(\frac{r_i - b}{t}\right), \qquad \sigma(u) = \begin{cases} \frac{1}{2}e^u, & u \le 0, \\ 1 - \frac{1}{2}e^{-u}, & u > 0, \end{cases} \tag{2}$$

where σ is the Laplace CDF. Each summand is continuous and strictly decreasing in b, so Ψ_r is a strictly decreasing bijection onto $(0, n)$ and the *budget equation* $\Psi_r(b) = k$ has a unique root for every $k \in (0, n)$, which we will simply denote by b. The LapSum operator is

$$\mathrm{LapSum}_k : \mathbb{R}^n \to (0,1)^n, \quad \mathrm{LapSum}_k(r)[i] = p_i = \sigma\left(\frac{r_i - b}{t}\right), \quad \sum_{i=1}^{n} p_i = k, \tag{3}$$

so a score far above b receives $p_i \approx 1$, one far below it $p_i \approx 0$, one at the threshold $p_i = \frac{1}{2}$. Since σ is the CDF of $\mathrm{Laplace}(0, t)$, the same object may be written as

$$p_i = \Pr\big(r_i + \varepsilon \ge b\big), \ \varepsilon \sim \mathrm{Laplace}(0, t), \qquad \Psi_r(b) = n \Pr(Y \ge b), \ Y = r_I + \varepsilon, \tag{4}$$

with random variable I uniform on $\{1, \dots, n\}$ independent of ε, which identifies b as a quantile of Y and is what the large-n solver of Sec. 4.2 exploits.

The constraint $\sum_i p_i = k$ with $p \in (0,1)^n$ gives the mask total mass k, which need not be an integer. That property holds only as $t \to 0$. We use *budget-exact* for the normalization and *exact* for the numerical solution of the budget equation. When a k-sparse support is required, the forward pass uses hard top-k and the backward pass uses LapSum_k as the straight-through surrogate.

Differentiating the implicit definition $\Psi_r(b) = k$ gives $\partial b / \partial r$ in closed form, and (Struski et al., 2025) obtains the vector–Jacobian product

$$(\nabla_r p)^\top g = f \odot (g - \langle g, q \rangle), \qquad q_i = \frac{f_i}{\sum_j f_j}, \qquad f_i = \frac{1}{2t} e^{-|b - r_i|/t}. \tag{5}$$

Once b is computed, evaluating this derivative costs only one element-wise pass and one reduction. We reuse this form unchanged in all variants. The entire computational problem is therefore the forward determination of b, the sole target of Sec. 4.

Zero-temperature limit and the relation to DFTopK. DFTopK fixes its threshold at the midpoint of the two boundary order statistics at every temperature, whereas LapSum solves for it. The following elementary statement, proved in Sec. A, specifies the limiting relation.

Proposition 1 (Zero-temperature limit of the LapSum threshold)**.** *Let $r \in \mathbb{R}^n$ have distinct entries with order statistics $r_{(1)} > \cdots > r_{(n)}$, while $k \in \{1, \ldots, n-1\}$, and let b be the unique root of $\Psi_r(b) = k$ at temperature $t > 0$. Then $b \to c := \frac{1}{2}(r_{(k)} + r_{(k+1)})$ and $\mathrm{LapSum}_k(r)[i] \to \mathbf{1}[r_i > c]$ as $t \to 0^+$, and the limiting mask has exactly k ones and $n-k$ zeros. Since c is by definition the DFTopK threshold, the LapSum threshold converges to it.*

This result concerns thresholds in the limit and does not establish equivalence of the operators. At fixed $t > 0$, the masks differ whenever $b \neq c$. The threshold b moves away from the midpoint to account for the non-boundary scores in the budget equation. This displacement preserves $\sum_i p_i = k$ and lets every score receive a gradient through the threshold in Eq. (5). In DFTopK, the threshold gradient reaches only $r_{(k)}$ and $r_{(k+1)}$. Tab. 10 measures the resulting budget error.

4 Fast LapSum: Solving the Budget Equation

The LapSum operator reduces to one scalar task: find the root b of $\Psi_r(b) = k$. Fast LapSum dispatches between two branches with the same output based on the size of the problem being solved. Sec. 4.3 states the guarantee for each branch, pseudocode for the full-sort and sample-bracketed branches is provided in Appendices B and C, respectively.

4.1 Exact Solve After Sorting

Let $s_1 \geq \cdots \geq s_n$ denote the scores sorted in descending order. Evaluating Ψ_r at a node s_i partitions the summation into scores strictly above s_i, the score itself, and scores strictly below it. Applying the Laplace CDF transforms each partition into a geometric-type sum, $\Psi_r(s_i) = (i-1) + \frac{1}{2}(1 + B_i - A_i)$, where $A_i = \sum_{j<i} e^{-(s_j - s_i)/t}$ and $B_i = \sum_{j>i} e^{-(s_i - s_j)/t}$. Because all exponents are non-positive, computing these sums is numerically stable and avoids the need for log-sum-exp stabilization. Furthermore, they satisfy the affine recurrences, $A_i = w_i(A_{i-1} + 1)$ and $B_i = v_i(B_{i+1} + 1)$ with $w_i = e^{-(s_{i-1} - s_i)/t}$, $v_i = e^{-(s_i - s_{i+1})/t}$ and $A_1 = B_n = 0$. Since affine recurrences are associative when expressed through a 2×2 transition matrix, a single cooperative block scan computes all node values efficiently. Because $\Psi_r(s_i)$ monotonically increases with i, the index lookup $i^\star = \min\{i : \Psi_r(s_i) \geq k\}$ brackets the root between two consecutive scores. Within this interval, the budget equation reduces to a quadratic polynomial in $m = e^{(b - s_{i^\star})/t}$,

$$A_{i^\star} m^2 - 2\big(i^\star - 1 - k\big)m - \big(1 + B_{i^\star}\big) = 0, \qquad b = s_{i^\star} + t \log m_+, \tag{6}$$

where m_+ denotes its unique positive root. Sec. B details the derivation, the cancellation-safe formulation, and the handling of tied scores. The bi-exponential structure underlying Eq. (6) is intrinsic to the LapSum formulation. By materializing the entire budget curve via a single parallel scan, our solver eliminates the need for iterative search procedures. Consequently, the overall algorithmic cost is reduced to just one sort, one scan, one lookup, one quadratic root calculation, and one fused CDF evaluation.

4.2 Probabilistic Bracketing: Localizing the Threshold

At the million scale, a full sort dominates computational cost and is inherently wasteful, as scores far from the threshold b require no internal ordering. We separate the localization of b from its exact determination by sampling kernel-noised scores. This allows us to stream the vector once, retaining only an ambiguous middle set $M = \{r_i : a_L \leq r_i < a_R\}$ to be solved exactly by the method in Sec. 4.1. Therefore, sampling determines only *where* the exact computation occurs. Prof-K (Dziarmaga et al., 2026) is the hard-selection counterpart of this structure, and Sec. D.2 compares the two methods. Provided $b \in [a_L, a_R)$, the scores outside the bracket remain on fixed

branches of the Laplace CDF. Their contributions collapse into stable exponential sums, $\tilde{S}_L = \sum_{r_i<a_L} e^{(a_L-r_i)/\alpha}$ and $\tilde{S}_R = \sum_{r_i\geq a_R} e^{(r_i-a_R)/\alpha}$, with non-positive exponents. Because these tail terms are exponentials in b sharing the rates of Eq. (8), they fold directly into A and B, preserving the quadratic interval root (Eq. (6)) over the restricted sub-interval (Sec. C).

This bracketing leverages the quantile interpretation of Eq. (4): $\Psi_r(b) = k$ means b is the q-quantile of the kernel-noised mixture Y, where $q = (n-k)/n$. By drawing K samples $y_s = r_{i_s} + \varepsilon_s$, we define the bracket using order statistics $a_L = y_{(j)}$ and $a_R = y_{(l)}$. While the exact solver processes scores in descending order to accumulate top-k masses, the bracketing algorithm sorts samples in ascending order to compute standard order statistics for quantile estimation. Because each indicator $\mathbf{1}[y_s \leq b]$ is Bernoulli(q), the count $C = \#\{s : y_s \leq b\}$ is Binomial(K, q), yielding the exact finite-sample coverage probability:

$$\Pr\big(a_L \leq b < a_R\big) = \Pr\big(j \leq C \leq l-1\big), \qquad C \sim \text{Binomial}(K, q), \tag{7}$$

This establishes Theorem 1. When both indices lie inside the sample range, we select j and l through a normal approximation with $z = 5.16$, giving coverage $1 - 2\Phi(-z) + \mathcal{O}(K^{-1/2})$ under i.i.d. sampling. Sec. D provides the exact finite-sample identity and a distribution-free Bernstein bound. The approximation does not apply when either index leaves the sample range. Algorithmic correctness remains independent of this approximation because certification detects a missed bracket.

For efficiency, the deployed fast path replaces i.i.d. sampling with systematic sampling, taking a strided view of the score buffer and adding pre-computed kernel noise. Although this sampler is not covered by Eq. (7), correctness remains independent of the sampler, see Sec. D.1. Certification requires one $\mathcal{O}(n)$ pass to compute $\Psi_r(a_L)$ and $\Psi_r(a_R)$ exactly, verifying $\Psi_r(a_R) \leq k \leq \Psi_r(a_L)$ without distributional assumptions. On failure, the margin is doubled, and the attempt is repeated. After four failed attempts, the implementation falls back to a full sort. While Prop. 3 establishes expected $\mathcal{O}(n)$ cost for an analyzed variant that draws i.i.d. samples on each retry and uses $K = \lceil n^{2/3} \rceil$, it does not establish this bound for the deployed systematic sampler, which is evaluated empirically. However, we demonstrate robust linear scaling in practice. We bound $|M|$ by relating the empirical law of r to Y through the Laplace kernel, as presented in Lemma 1. Crucially, certification and exactness are strictly enforced by an $\mathcal{O}(n)$ verification pass, which bounds the worst-case cost to $\mathcal{O}(\log N)$ complexity, allowing the fast path to operate safely.

4.3 What Is Guaranteed, and by Which Path

The words *exact*, *linear-time* and *fast path* refer to three properties under distinct guarantees. **(P1) Budget exactness** is deterministic when the bracket contains b, certification accepts it, or when the full-data fallback is used. The resulting mask satisfies $\sum_i p_i = k$ up to floating-point precision (Tab. 10). **(P2) Threshold localization** is based on probabilistic bounds for i.i.d. sampling, as in Eq. (7), while Sec. 6.3 measures the deployed systematic sampler. Exact certification strictly enforces deterministic acceptance by checking both endpoints. **(P3) Linear running time** is guaranteed in expectation for the i.i.d. variant Prop. 3. In practice, the deployed systematic sampler achieves strictly linear empirical scaling, bounded by a worst-case $\mathcal{O}(n \log n)$ full-sort fallback. Importantly, all reported benchmarks and downstream experiments (Sec. 6) run the fully certified solver, actively accounting for the cost of endpoint verification and any necessary retries.

The certified implementation scales across three additional orders of magnitude, as reported in Tab. 8 in Sec. F. At $N{=}10^6$, the full-sort branch takes 6.392 ms and the certified path takes 0.923 ms, giving a $6.9\times$ speedup. At $N{=}10^8$, the corresponding times are 1190.9 and 7.241 ms, giving a $164.5\times$ speedup. The certified $N{=}10^8$ runtime has a 95th percentile of 15.576 ms and a retry rate of 33.3%.

5 Speed: Fast LapSum at GPU-Primitive Cost

All benchmarks run on a single NVIDIA GH200 120GB using PyTorch 2.9.1 and CUDA 12.9 in FP32, reporting median CUDA-event times after warm-up, see details in Sec. E. For certified Fast LapSum, we use 30 repetitions through $N = 10^7$, 24 repetitions at $3{\times}10^7$ and 10^8, and 20 repetitions at $3{\times}10^8$ and 10^9. The forward–backward measurements use 30 repetitions at every reported size. The remaining operator curves use 20 repetitions below 10^6, eight from 10^6 to below 10^7, and five at larger sizes. Tab. 6 isolates our algorithmic gains from CUDA-specific optimizations: in

B=16, n=4096, replacing the 60-step bisection with the full scan gives a $11.8\times$ speedup. Cooperative execution and CDF fusion increase the total speedup to $72.6\times$. Furthermore, our analytical vector–Jacobian product matches centered finite differences in FP64 with a relative error below 2.4×10^{-10}, compare Tab. 7.

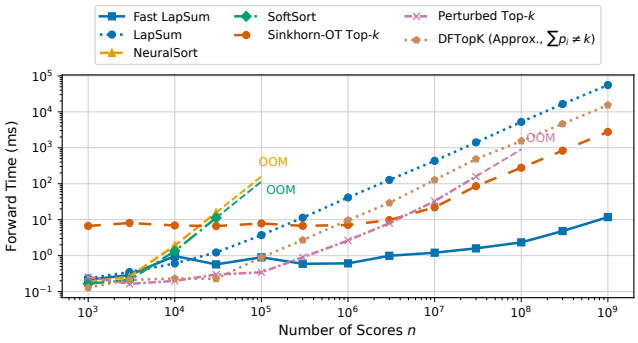

Figure 2: Certified forward runtime on an NVIDIA GH200 120GB (B=1, k=$N/16$, FP32). Fast LapSum timings include the endpoint certificate and any retries. Dense sort-matrix methods exceed device memory at scale. At $N = 10^8$, certified Fast LapSum has a median runtime of 7.24 ms.

Against Differentiable-Ordering Operators. Fig. 2 compares certified Fast LapSum with the vanilla LapSum and other differentiable-ordering relaxations in FP32. At 10^8 and 10^9 scores, Fast LapSum takes 7.24 and 121.76 ms, while LapSum takes 5240.81 and 55591.47 ms. These values correspond to speedups of $724\times$ and $457\times$, respectively. NeuralSort and SoftSort require quadratic score matrices whose minimum live-tensor estimates exceed GH200 memory from $n = 10^5$, while Perturbed Top-k runs out of memory from $n = 10^8$. These endpoints are hardware-dependent. Hard top-k is not plotted because it does not provide the soft exact-budget gradient used in our applications.

Budget Accuracy. Tab. 10 in Sec. F evaluates the relative budget error $|\sum_i p_i - k|/k$. Fast LapSum maintains exact normalization to FP32 precision across all scales. In contrast, DFTopK's fixed midpoint violates the budget equation with error 2.46, yielding mass $\approx 3.46k$, while other exact-budget relaxations are either out of memory or two orders of magnitude slower. Since competing operators target different objectives, this comparison serves strictly as a compatibility check for the $\sum_i p_i = k$ normalization required by our pipelines, not a general quality ranking.

Forward-backward timings exhibit identical scaling behaviour, as the backward pass simply evaluates the analytical VJP from Equation (5).

6 Experiments

6.1 Learned Pruning for 3D Gaussian Splatting

Protocol. We add a learned pruning stage to the standard 3D Gaussian Splatting Adaptive Density Control (ADC) (Kerbl et al., 2023). ADC creates the candidate Gaussians until iteration $10,000$. Then, densification stops, one score per Gaussian is optimized through an exact-budget soft mask, and the highest-scoring K Gaussians are retained at iteration $12,000$. Training of Gaussian parameters continues on this hard set until iteration $30,000$. We compare vanilla ADC, ADC with Fast LapSum, and a matched hard-pruning control that ranks Gaussians by footprint. Both pruning methods retain the same K and receive the same recovery period. The main evaluation covers all 13 scenes from Mip-NeRF 360, Tanks & Temples, and Deep Blending with three seeds. Implementation details and the score-logging protocol are in Sec. G.

Choosing the retention rate. Before the full evaluation, we select a single global operating point using a separate five-scene and one seed. Fig. 3 shows the resulting quality–efficiency trade-off as the pruning rate increases. At 70% pruning, the mean PSNR change relative to full ADC is -0.065 dB; at 90% pruning, it decreases to -0.402 dB. Based on this trade-off, we set 30% retention for all subsequent 13-scene and large-scene evaluations.

13-scene evaluation. Using the 30% retention rate selected above, we evaluate Fast LapSum on all 13 scenes with three seeds per scene. Tab. 1 reports the quality change relative to the vanilla ADC, and the remaining number of Gaussians, training time, rendering throughput, and model size.

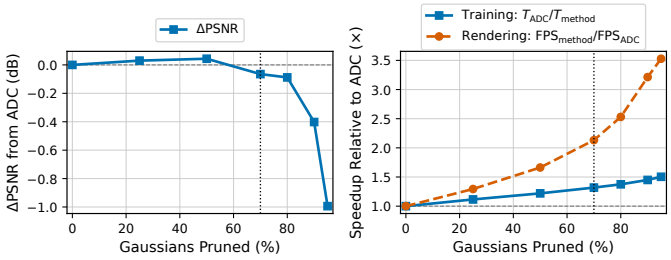

Figure 3: Retention sweep averaged across five scenes with one seed. Left: mean PSNR change compared to the vanilla ADC. Right: training $T_{\mathrm{ADC}}/T_{\mathrm{method}}$ and rendering $\mathrm{FPS}_{\mathrm{method}}/\mathrm{FPS}_{\mathrm{ADC}}$ speedups. The dotted line marks the selected operating point of 70% pruning (30% retention).

Table 1: Quality change and resource usage compared to the vanilla ADC at 30% retention. We first average the three seeds for each scene, then compute the relative difference, and report the mean ± standard deviation over 13 scenes. A small magnitude indicates that pruning preserves ADC quality.

Method	ΔPSNR (dB) ↑	ΔSSIM ↑	ΔLPIPS ↓	N (M) ↓	Train (min) ↓	FPS ↑	Model (MiB) ↓
ADC (reference)	–	–	–	2.52 ± 1.34	12.6 ± 2.6	201.9 ± 43.5	595.2 ± 317.9
+hard pruning	-0.26 ± 0.19	-0.009 ± 0.006	0.022 ± 0.010	$\mathbf{0.76 \pm 0.40}$	10.3 ± 2.4	310.3 ± 63.5	$\mathbf{178.5 \pm 95.2}$
+Fast LapSum	-0.05 ± 0.12	-0.002 ± 0.003	0.012 ± 0.008	$\mathbf{0.76 \pm 0.40}$	$\mathbf{10.0 \pm 2.3}$	$\mathbf{385.9 \pm 57.3}$	$\mathbf{178.5 \pm 95.2}$

Table 2: Large-scene quality and efficiency at 30% retention. Quality changes are paired relative to the matched ADC baseline. Results are means ± standard deviations over three seeds. Model size (MiB) scales linearly with N and is omitted for brevity. Bold marks the best mean for each scene.

Scene	Method	ΔPSNR ↑	ΔSSIM ↑	ΔLPIPS ↓	N (M) ↓	Train (min) ↓	FPS ↑
Mill-19 Bldg.	ADC (reference)	–	–	–	9.51 ± 0.12	56.3 ± 0.3	73.2 ± 4.9
	+ Fast LapSum	-0.17 ± 0.11	-0.012 ± 0.001	0.017 ± 0.000	$\mathbf{2.85 \pm 0.04}$	$\mathbf{46.7 \pm 0.3}$	$\mathbf{198.4 \pm 2.0}$
	+ LapSum	-0.14 ± 0.05	-0.010 ± 0.000	0.016 ± 0.000	$\mathbf{2.85 \pm 0.04}$	91.9 ± 0.7	197.3 ± 1.3
Rubble (10M)	ADC (reference)	–	–	–	10.74 ± 0.03	46.7 ± 0.5	105.7 ± 1.0
	+ Fast LapSum	-0.25 ± 0.05	-0.016 ± 0.000	0.026 ± 0.000	$\mathbf{3.22 \pm 0.01}$	$\mathbf{38.1 \pm 0.0}$	236.3 ± 4.4
	+ LapSum	-0.26 ± 0.11	-0.015 ± 0.001	0.026 ± 0.000	$\mathbf{3.22 \pm 0.01}$	89.7 ± 0.5	$\mathbf{237.9 \pm 2.3}$
Rubble (28M)	ADC (reference)	–	–	–	27.73 ± 0.33	80.4 ± 0.4	49.9 ± 4.1
	+ Fast LapSum	$+0.06 \pm 0.15$	-0.002 ± 0.001	0.009 ± 0.001	$\mathbf{8.32 \pm 0.10}$	$\mathbf{61.0 \pm 0.1}$	$\mathbf{136.5 \pm 0.5}$
	+ LapSum	$+0.09 \pm 0.10$	-0.002 ± 0.001	0.009 ± 0.001	$\mathbf{8.32 \pm 0.10}$	211.2 ± 2.2	136.3 ± 0.8

At 30% retention, Fast LapSum only slightly changes mean PSNR, SSIM, and LPIPS by -0.046 dB, -0.0020, and $+0.0117$ relative to the vanilla ADC, despite a significant reduction in the final number of Gaussians, stored model size by 70%, shortening training time from 12.59 to 9.97 minutes ($1.26\times$), and increasing rendering throughput from 201.9 to 385.9 FPS ($1.91\times$). Compared with hard pruning, Fast LapSum improves all evaluation metrics. The example of the reconstructed scene by Fast LapSum and ADC is presented in Fig. 1.

Large-scene evaluation. We next evaluate Fast LapSum on substantially larger 3DGS scenes, where the number of Gaussians reaches tens of millions of elements. This setting verifies the practical benefit of replacing the original LapSum solver with Fast LapSum inside the learned pruning pipeline in demanding conditions.

ADC grows the number of 3D primitives until iteration $30,000$. Score learning continues until iteration $36,000$, when the K highest-scoring Gaussians are retained, and training then continues on the pruned set until iteration $60,000$. In the Rubble scene, the number of Gaussians reaches 10.7 million in the 10M configuration and 27.6–28.1 million in the 28M configuration, while for Mill-19 Building it is about 9.5 million.

Across the three configurations, 30% retention gives a 2.24–$2.74\times$ rendering speedup and a 1.21–$1.32\times$ training speedup over ADC, while PSNR changes the range from -0.25 to $+0.06$ dB. Compared to LapSum, Fast LapSum reduces the end-to-end training time by 1.97–$3.47\times$ while maintaining the same final number of primitives and closely matched quality. In the case of memory

size, Fast LapSum-based model needs around 1967.6 and 674.9 MiB for Rubble (28M) and Mill-19 Building, respectively, while for ADC these values are much larger (6558.5 and 2249.7 MiB).

6.2 Exact-Budget Adversarial Examples

Table 3: ImageNet attack success rate (%). Bold marks the best method at each exact budget.

	ConvNeXt			ResNet-50		
Method	K=10	100	500	K=10	100	500
Fast LapSum	**11.9**	**78.6**	**100.0**	**24.5**	**92.6**	**100.0**
Sparse-PGD	7.2	47.0	94.6	15.5	66.3	98.2
PGD-L_0	2.3	20.3	82.5	4.8	48.2	91.8

Threat model and protocol. We study untargeted white-box pixel-L_0 adversarial attacks, where changing any combination of RGB channels at a single spatial location counts as a single perturbation. The forward pass always uses an exact hard top-K mask, with Fast LapSum supplying the score gradient through the straight-through estimator. We optimize both the mask scores and replacement colors using a Carlini–Wagner margin loss for at most 500 steps and three restarts. Finally, attack success is measured only after the adversarial images are discretized to standard 8-bit color depth. Our primary ImageNet evaluation uses 1,000 labeled validation images classified correctly by both ConvNeXt-V2 Tiny (Woo et al., 2023) and ResNet-50. Sparse-PGD (Zhong & Liu, 2025) and PGD-L_0 (Croce & Hein, 2019) receive the same fixed support budget and optimization limit.

Fast LapSum achieves the highest measured attack success rate at every reported budget on both classifiers. At $K = 100$, the success rates are 78.6% on ConvNeXt and 92.6% on ResNet-50, which is presented in Tab. 3. The corresponding rates are only 47.0% and 66.3% for Sparse-PGD, and 20.3% and 48.2% for PGD-L_0. The mean time for one image and one restart includes optimization and all model evaluations until success or the 500-step limit. On ConvNeXt, these times are 3.31, 6.17, and 10.45 seconds for Fast LapSum, Sparse-PGD, and PGD-L_0. On ResNet-50, they are 1.26, 3.11, and 5.83 seconds, respectively.

Table 4: Native-resolution DIV2K comparison (10 images, 3 restarts each). ASR is the fraction of images attacked successfully in at least one restart. Total time is the mean duration of one image-restart attack. Selector time is the cumulative time spent in performing differentiable selection operations. An attack stops either after success or 500 optimization steps. Bold marks the lower time per budget.

		Total time (s) ↓		Selector time (s) ↓	
K	ASR, both (%)	Fast LapSum	LapSum	Fast LapSum	LapSum
100	0	**30.29**	61.60	**2.16**	27.31
500	0	**30.41**	72.13	**2.28**	37.89
1000	40	**25.36**	57.18	**2.10**	29.49
1500	60	**17.95**	40.20	**2.05**	18.50
2000	80	**13.13**	24.81	**1.76**	11.18

Native-resolution DIV2K. To evaluate our approach at millions of coordinates, we selected ten native-resolution DIV2K images with 2.35–2.99 million pixels. We generate clean top-1 pseudolabels using a frozen ConvNeXt-V2, filter for confidences ≥ 0.8, and perform untargetted attacks against these labels. Both Fast LapSum and the baseline LapSum share identical initializations, losses, schedules, and forward-pass hard masks, differing only in the backward-pass soft selector.

Because they implement the identical mathematical operator via different threshold solvers, both methods achieve matching attack success sets at every budget. However, Fast LapSum is 1.89–$2.37\times$ faster end-to-end and 6.35–$16.63\times$ faster in selector time, compare Tab. 4. For context on sparsity, 1,500-pixels correspond to only 0.050–0.064% of the image pixels.

6.3 Bracketing Robustness

We evaluate the probabilistic bracketing mechanism independently of downstream tasks across a comprehensive grid of vector sizes, i.e. $n \in \{10^5, 10^6, 10^7\}$, retention ratios $1/2$, $1/16$, and $1/100$, and three score-to-temperature scales. We benchmark the deployed systematic sampler against i.i.d. sampling with replacement and sampling without replacement across ten synthetic and real-world distributions: Gaussian, Student-t_2, log-normal, Cauchy, a three-cluster mixture, a 16-level tied distribution, Bernoulli, standardized Binomial, learned 3DGS scores and budget-matched bimodal. For each configuration, trials generate a score vector, compute the exact threshold through a full sort, and assess the initial and validated-bracket correctness.

On a calibration grid of $z \in \{1, 1.96, 3, 5.16\}$, the i.i.d. sampler yields $69.7\%, 95.5\%, 99.8\%$, and 100% initial coverage, while the systematic sampler achieves $63.6\%, 90.8\%, 99.4\%$, and 100%, as

Table 5: Large-scale evaluation of the systematic sampler (n=10^7, k=$n/16$, z=5.16, eight trials per row). Coverage is the fraction of initial brackets containing the full-sort threshold. Max. budget error is the post-validation value $\max_{\text{trials}} |\sum_i p_i - k|/k$. Retry rate is omitted because it equals $100\% -$ Coverage. Bold marks the highest coverage.

Distribution	Coverage (%)	Max. budget error	Distribution	Coverage (%)	Max. budget error
Gaussian	75.0	1.38×10^{-7}	16-level ties	**100.0**	2.52×10^{-5}
Student-t_2	87.5	1.18×10^{-7}	Bernoulli	87.5	7.95×10^{-5}
Log-normal	**100.0**	4.42×10^{-8}	Binomial(16, 0.25)	**100.0**	4.22×10^{-5}
Cauchy	62.5	7.73×10^{-6}	Learned 3DGS	**100.0**	1.61×10^{-7}
Three clusters	**100.0**	1.32×10^{-7}	Bimodal (budget-matched)	**100.0**	5.67×10^{-10}

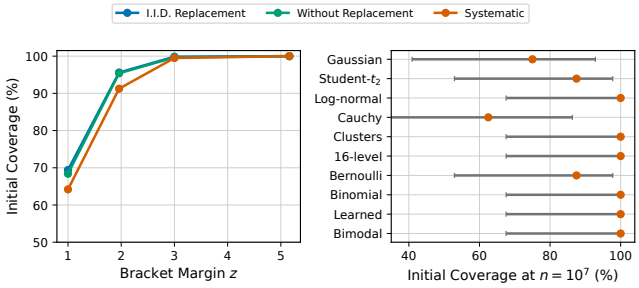

Figure 4: Initial bracket coverage before validation. Left: calibration at $n = 10^5$ across ten distributions with 1,800 trials per sampler and margin. Right: the deployed systematic sampler at $n = 10^7$, $k = n/16$, unit score-to-temperature scale, and $z = 5.16$. Error bars are Wilson 95% confidence intervals over eight trials.

presented in Fig. 4. The maximum validated relative budget error was 7.95×10^{-5}, and none of the trials required full-sort fallback. In practice, after some iterations, scores often form a bimodal distribution of suppressed and enhanced scores. Although the systematic sampler lacks i.i.d. guarantees, slightly lowering coverage at small z, exact validation ensures correctness. As Tab. 5 demonstrates, the systematic sampler achieves perfect coverage in them. Across all ten distributions at $n = 10^7$, the initial systematic bracket succeeds in 91.25% of trials (73/80). The remaining seven cases are resolved by bracket widening, and no trial invokes the full-sort fallback. Thus, in practice, the deployed solver usually remains in its linear-work bracketed regime.

7 Conclusion

We presented Fast LapSum, a GPU-native solver for exact-budget differentiable top-k selection. The i.i.d. retry variant analyzed in Prop. 3 has expected linear running time under its stated regularity assumptions. The deployed systematic implementation has empirical scaling, while certification and the exact fallback preserve correctness. At 10^9 dimensions, the certified implementation is $457\times$ faster than LapSum. As a drop-in replacement, Fast LapSum accelerates megapixel adversarial attacks and enables massive-scale 3D Gaussian Splatting scene pruning, yielding faster and lightweight rendering with negligible quality loss. Ultimately, these results demonstrate that exact-budget differentiable top-k selection is practical for modern, million-parameter machine learning pipelines.

7.1 Limitations

The construction of our operator is tied to the Laplace kernel. The expected-linear result applies only to the i.i.d. retry variant under assumptions (H1)–(H4), including an interior target quantile. It does not cover the deployed systematic sampler or fixed and extremely small budgets as n grows, while some experiments evaluate the latter regime empirically using the certified implementation. Certification and exact fallback preserve correctness, but runtime may approach the $\mathcal{O}(n \log n)$ full-sort solver in the worst-case scenario. Heavy ties can enlarge the middle band. Memory limits and branch crossovers remain hardware-specific.

AI USE STATEMENT

In this work, we have not used generative AI tools for formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, or interpreting results. Furthermore, generating synthetic data sets, assisting with translation, cleaning and reformatting datasets, and supporting qualitative and thematic data analysis are not applicable to this work.

We used generative AI tools for helping develop theoretical models or conceptual frameworks, implementing methods, discovering research topics, identifying relevant literature, summarizing existing literature, and editing the research paper to improve readability. We have reviewed all AI-assisted work. Specifically, any AI-generated code or method implementation was rigorously tested, executed, and manually verified for correctness by the authors. Additionally, all AI-suggested literature was manually verified to ensure the sources exist and accurately reflect the cited content, and all AI-edited text was manually reviewed, refined, and approved by at least two authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The exact solve is specified by Eqs. (6) and (8), with the cancellation-safe root and tie handling in Sec. B. The bracketed solve is given by Eqs. (14) and (15). Both branches appear as pseudocode in Supplementary Materials. The analytical VJP in Eq. (5) is checked against finite differences in Tab. 7. Secs. E, F and F.1 provide implementation and benchmark details. Sec. G specifies the 3DGS and adversarial-attack protocols.

REFERENCES

Tolu Alabi, Jeffrey D. Blanchard, Bradley Gordon, and Russel Steinbach. Fast k-selection algorithms for graphics processing units. In *ACM Journal of Experimental Algorithmics*, 2012.

Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis Bach. Learning with differentiable perturbed optimizers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. URL https://arxiv.org/abs/2002.08676.

Mathieu Blondel, Olivier Teboul, Quentin Berthet, and Josip Djolonga. Fast differentiable sorting and ranking. In *International Conference on Machine Learning (ICML)*, 2020. URL https://arxiv.org/abs/2002.08871.

Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973.

Francesco Croce and Matthias Hein. Sparse and imperceivable adversarial attacks. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019. URL https://arxiv.org/abs/1909.05040.

Tadeusz Dziarmaga, Witold Sikora, Łukasz Struski, Jacek Tabor, and Marcin Mazur. Prof-k: Probabilistic one-pass filtering for efficient top-k selection. *arXiv preprint arXiv:2608.12573*, 2026.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research (JMLR)*, 2022. URL https://arxiv.org/abs/2101.03961.

Robert W. Floyd and Ronald L. Rivest. Expected time bounds for selection. *Communications of the ACM*, 18(3):165–172, 1975.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://arxiv.org/abs/1803.03635.

Leo Gao, Tom Dupré la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. Scaling and evaluating sparse autoencoders. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2406.04093.

Aditya Grover, Eric Wang, Aaron Zweig, and Stefano Ermon. Stochastic optimization of sorting networks via continuous relaxations. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://arxiv.org/abs/1903.08850.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015. URL https://arxiv.org/abs/1506.02626.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, George Drettakis, et al. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1, 2023.

Alireza Makhzani and Brendan Frey. k-sparse autoencoders. In *International Conference on Learning Representations (ICLR)*, 2014. URL https://arxiv.org/abs/1312.5663.

Felix Petersen, Christian Borgelt, Hilde Kuehne, and Oliver Deussen. Differentiable sorting networks for scalable sorting and ranking supervision. In *International Conference on Machine Learning (ICML)*, 2021. URL https://arxiv.org/abs/2105.04019.

Sebastian Prillo and Julian Martin Eisenschlos. SoftSort: A continuous relaxation for the argsort operator. In *International Conference on Machine Learning (ICML)*, 2020. URL https://arxiv.org/abs/2006.16038.

Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. DynamicViT: Efficient vision transformers with dynamic token sparsification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. URL https://arxiv.org/abs/2106.02034.

Michael E. Sander, Joan Puigcerver, Josip Djolonga, Gabriel Peyré, and Mathieu Blondel. Fast, differentiable and sparse top-k: a convex analysis perspective. In *International Conference on Machine Learning (ICML)*, 2023. URL https://arxiv.org/abs/2302.01425.

Victor Sanh, Thomas Wolf, and Alexander M. Rush. Movement pruning: Adaptive sparsity by fine-tuning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. URL https://arxiv.org/abs/2005.07683.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations (ICLR)*, 2017. URL https://arxiv.org/abs/1701.06538.

Łukasz Struski, Michal B. Bednarczyk, Igor T. Podolak, and Jacek Tabor. LapSum - one method to differentiate them all: Ranking, sorting and top-k selection. In *Proceedings of the 42nd International Conference on Machine Learning*, pp. 56990–57007, 2025.

Sanghyun Woo, Shoubhik Debnath, Ronghang Hu, Xinlei Chen, Zhuang Liu, In So Kweon, and Saining Xie. ConvNeXt V2: Co-designing and scaling convnets with masked autoencoders. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. URL https://arxiv.org/abs/2301.00808.

Yujia Xie, Hanjun Dai, Minshuo Chen, Bo Dai, Tuo Zhao, Hongyuan Zha, Wei Wei, and Tomas Pfister. Differentiable top-k operator with optimal transport. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. URL https://arxiv.org/abs/2002.06504.

Xuyang Zhong and Chen Liu. Sparse-pgd: A unified framework for sparse adversarial perturbations generation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2025.

Yanjie Zhu, Zhen Zhang, Yunli Wang, Zhiqiang Wang, Yu Li, Rufan Zhou, Shiyang Wen, Peng Jiang, Chenhao Lin, and Jian Yang. Differentiable fast top-k selection for large-scale recommendation, 2025. URL https://arxiv.org/abs/2510.11472.

APPENDIX

A PROOF OF THE ZERO-TEMPERATURE LIMIT

We prove Prop. 1. To make the temperature explicit, write Ψ_r for the budget curve in Eq. (2) and b_t for its unique root.

Proposition 2. *Let $r \in \mathbb{R}^n$ have distinct entries with order statistics $r_{(1)} > \cdots > r_{(n)}$, let $k \in \{1, \ldots, n-1\}$, and let b_t be the unique solution of $\Psi_t(b) = k$. Define*

$$c = \frac{r_{(k)} + r_{(k+1)}}{2}, \qquad d = \frac{r_{(k)} - r_{(k+1)}}{2} > 0,$$

and

$$p_i(t) = \sigma\left(\frac{r_i - b_t}{t}\right).$$

Then $b_t \to c$ and $p_i(t) \to \mathbf{1}[r_i > c]$ as $t \to 0^+$. Hence the limiting mask has exactly k ones.

Proof. Fix $\varepsilon \in (0, d)$ and define

$$\lambda_t = e^{-(d-\varepsilon)/t}, \qquad \mu_t = e^{-(d+\varepsilon)/t}.$$

Then

$$\frac{\mu_t}{\lambda_t} = e^{-2\varepsilon/t} \to 0,$$

so for all sufficiently small t we have $\mu_t < \lambda_t/n$.

Consider first $b = c - \varepsilon$. For the two boundary scores,

$$r_{(k)} - b = d + \varepsilon, \qquad r_{(k+1)} - b = -(d - \varepsilon),$$

and therefore their contributions to $\Psi_t(b)$ are

$$1 - \frac{1}{2}\mu_t \qquad \text{and} \qquad \frac{1}{2}\lambda_t,$$

respectively. For every $i < k$, $r_{(i)} - b > d + \varepsilon$, so its contribution is at least $1 - \frac{1}{2}\mu_t$. Hence

$$\Psi_t(c - \varepsilon) \geq k\left(1 - \frac{1}{2}\mu_t\right) + \frac{1}{2}\lambda_t = k + \frac{1}{2}(\lambda_t - k\mu_t) > k,$$

where the last inequality follows from $k < n$ and $\mu_t < \lambda_t/n$.

Now take $b = c + \varepsilon$. Then

$$r_{(k)} - b = d - \varepsilon, \qquad r_{(k+1)} - b = -(d + \varepsilon),$$

so the boundary contributions are

$$1 - \frac{1}{2}\lambda_t \qquad \text{and} \qquad \frac{1}{2}\mu_t.$$

The $k - 1$ scores above $r_{(k)}$ contribute at most 1 each, while every score below $r_{(k+1)}$ contributes at most $\frac{1}{2}\mu_t$. Therefore

$$\Psi_t(c + \varepsilon) \leq (k - 1) + \left(1 - \frac{1}{2}\lambda_t\right) + \frac{n-k}{2}\mu_t = k - \frac{1}{2}\big(\lambda_t - (n-k)\mu_t\big) < k.$$

Since Ψ_t is strictly decreasing and $\Psi_t(b_t) = k$, it follows that

$$c - \varepsilon < b_t < c + \varepsilon$$

for all sufficiently small t. As $\varepsilon \in (0, d)$ was arbitrary,

$$b_t \to c.$$

It remains to consider the mask. If $i \le k$, then

$$r_i - b_t \ge r_{(k)} - b_t \to d > 0,$$

and therefore

$$\frac{r_i - b_t}{t} \to +\infty.$$

Thus $p_i(t) \to 1$. Similarly, if $i \ge k+1$,

$$r_i - b_t \le r_{(k+1)} - b_t \to -d < 0,$$

so

$$\frac{r_i - b_t}{t} \to -\infty$$

and $p_i(t) \to 0$. Hence

$$p_i(t) \to \mathbf{1}[r_i > c].$$

Since c lies strictly between $r_{(k)}$ and $r_{(k+1)}$, this limiting mask has exactly k ones. □

B Derivation of the Full-Sort Solver

Algorithm 1 Fast LapSum: Exact Full-Sort Branch.

Require: scores $r \in \mathbb{R}^n$, budget $k \in (0, n)$, temperature $t > 0$

1: $s \leftarrow \textsc{SortDescending}(r)$ ▷ $\mathcal{O}(n \log n)$
2: $A \leftarrow \textsc{Scan}(w)$, $B \leftarrow \textsc{ReverseScan}(v)$ by Eq. (9) ▷ one cooperative block scan
3: $\Psi_i \leftarrow (i-1) + \frac{1}{2}(1 + B_i - A_i)$ for all i ▷ increasing in i
4: $i^\star \leftarrow \min\{i : \Psi_i \ge k\}$ ▷ binary search
5: $b \leftarrow s_{i^\star} + t \log m_+$ with m_+ the positive root of Eq. (6) ▷ closed form
6: **return** $p_i = \sigma\big((r_i - b)/t\big)$ for all i ▷ fused with the scan kernel

The full-sort solver evaluates the budget curve at the sorted scores and uses these values to locate the interval containing the threshold. Fig. 5 illustrates the underlying LapSum geometry. The two-exponential structure inside an interval is inherited from Struski et al. (2025); here we derive the scan form and the closed-form root used by Algorithm 1.

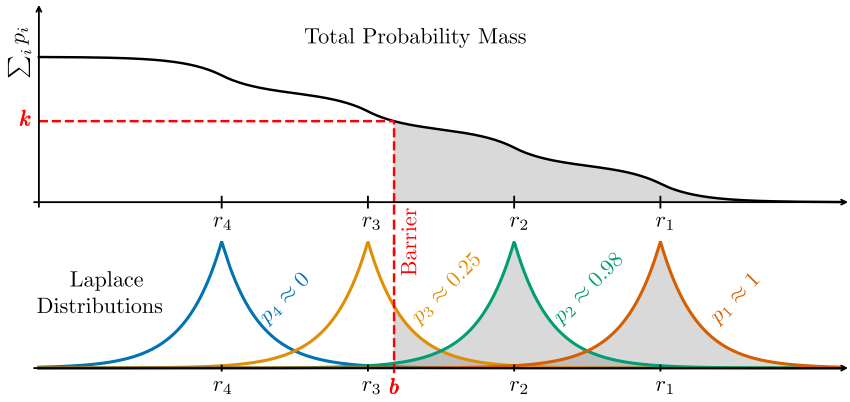

Figure 5: LapSum geometry. Each score r_i carries a Laplace kernel of width$t = |\alpha|$, and $p_i = \Pr(r_i + \varepsilon \ge b)$ is the mass above the threshold b. The threshold is chosen such that $\sum_i p_i = k$. As $t \to 0$, the soft mask converges to hard top-k.

Node values. Let $s_1 \ge s_2 \ge \cdots \ge s_n$ denote the scores sorted in descending order, and recall

$$\Psi_r(b) = \sum_{j=1}^{n} \sigma\left(\frac{s_j - b}{t}\right).$$

At $b = s_i$, the terms with $j < i$, $j = i$, and $j > i$ contribute $1 - \frac{1}{2}e^{-(s_j - s_i)/t}$, $\frac{1}{2}$, and $\frac{1}{2}e^{-(s_i - s_j)/t}$, respectively. Hence

$$\Psi_r(s_i) = (i-1) + \tfrac{1}{2}\big(1 + B_i - A_i\big), \quad A_i = \sum_{j<i} e^{-(s_j - s_i)/t}, \quad B_i = \sum_{j>i} e^{-(s_i - s_j)/t}. \tag{8}$$

The quantities A_i and B_i satisfy

$$A_i = w_i\,(A_{i-1} + 1), \quad w_i = e^{-(s_{i-1} - s_i)/t}, \qquad B_i = v_i\,(B_{i+1} + 1), \quad v_i = e^{-(s_i - s_{i+1})/t}, \tag{9}$$

with $A_1 = B_n = 0$. All exponents are non-positive, so the recurrences do not require log-sum-exp stabilization. Writing the update $x_i = w_i(x_{i-1} + 1)$ as

$$\begin{pmatrix} x_i \\ 1 \end{pmatrix} = \begin{pmatrix} w_i & w_i \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_{i-1} \\ 1 \end{pmatrix}$$

makes the composition associative. A parallel scan therefore computes all A_i, and a reverse scan computes all B_i. Since Ψ_r is decreasing in its argument and s_i is non-increasing in i, the sequence $\Psi_r(s_i)$ is non-decreasing. Thus

$$i^\star = \min\{i : \Psi_r(s_i) \geq k\}$$

locates the interval containing the root.

Interval root. Write

$$b = s_{i^\star} + t\delta, \qquad m = e^\delta \geq 1,$$

with $b \leq s_{i^\star - 1}$. For $j < i^\star$,

$$e^{-(s_j - b)/t} = m\, e^{-(s_j - s_{i^\star})/t},$$

whereas for $j \geq i^\star$,

$$e^{(s_j - b)/t} = m^{-1} e^{(s_j - s_{i^\star})/t}.$$

Substituting these terms into the budget equation gives

$$(i^\star - 1) - \tfrac{1}{2} m A_{i^\star} + \tfrac{1}{2m}\big(1 + B_{i^\star}\big) = k. \tag{10}$$

Multiplying by $2m$ gives Eq. (6) from the main text. Let $\Delta_\star = i^\star - 1 - k$. Since $A_{i^\star} \geq 0$ and $1 + B_{i^\star} > 0$, the quadratic has one positive root,

$$m_+ = \frac{\Delta_\star + \sqrt{\Delta_\star^2 + A_{i^\star}(1 + B_{i^\star})}}{A_{i^\star}} = \frac{1 + B_{i^\star}}{\sqrt{\Delta_\star^2 + A_{i^\star}(1 + B_{i^\star})} - \Delta_\star}. \tag{11}$$

The two expressions are algebraically identical. We use the first for $\Delta_\star \geq 0$ and the second for $\Delta_\star < 0$ to avoid subtractive cancellation. The second form also covers $A_{i^\star} = 0$, which occurs for $i^\star = 1$. Finally,

$$b = s_{i^\star} + t \log m_+.$$

Ties. Equal scores require no separate case. If $s_{i-1} = s_i$, then $w_i = 1$ in Eq. (9), and the corresponding node values $\Psi_r(s_i)$ are equal. The lookup therefore selects the first index satisfying $\Psi_r(s_i) \geq k$. The interval equation and the returned mask remain unchanged because the mask depends on each score only through $\sigma((r_i - b)/t)$.

C Exact Solve Inside a Bracket

Assume $b \in [a_L, a_R)$ and partition the scores into

$$L = \{i : r_i < a_L\}, \qquad M = \{i : a_L \leq r_i < a_R\}, \qquad R = \{i : r_i \geq a_R\}. \tag{12}$$

For every b in the bracket, the scores in L stay below b and those in R stay above it, so with the tail sums

$$\tilde{S}_L = \sum_{r_i < a_L} e^{(a_L - r_i)/\alpha}, \qquad \tilde{S}_R = \sum_{r_i \geq a_R} e^{(r_i - a_R)/\alpha}, \tag{13}$$

whose exponents are all non-positive,

$$\Psi_r(b) = |R| - \tfrac{1}{2}\tilde{S}_R\, e^{-(a_R - b)/t} + \tfrac{1}{2}\tilde{S}_L\, e^{-(b - a_L)/t} + \sum_{i \in M} \sigma\big((r_i - b)/t\big). \tag{14}$$

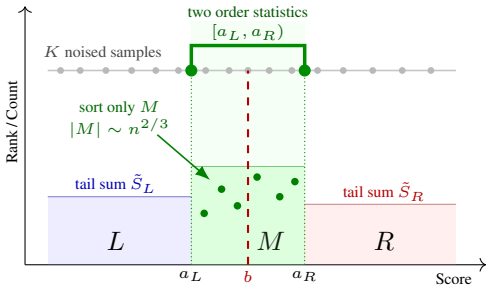

Figure 6: The sampled order statistics define thresholds a_L and a_R around the threshold b. These thresholds partition the samples into subsets L, M, and R. The tails L and R are aggregated into $\tilde{S}_L$ and $\tilde{S}_R$, whereas only the middle subset, $M = \{r_i : a_L \le r_i < a_R\}$, is sorted and used by the exact solver.

Algorithm 2 Fast LapSum: Sample-Bracketed Branch for large n.

Require: scores $r \in \mathbb{R}^n$, budget k, temperature $t > 0$, margin z, flag CERTIFY

$q \leftarrow (n-k)/n$; $K \leftarrow \lceil n^{2/3} \rceil$

draw K kernel-noised samples $y_s = r_{i_s} + \varepsilon_s$ and sort them ascending ▷ systematic in the fast path, i.i.d. when analyzed

$j, l \leftarrow$ binomial quantiles of $\mathrm{Binomial}(K, q)$ at level z, or Eq. (19)

if $j < 1$ or $l > K$ then return Algorithm 1

$a_L \leftarrow y_{(j)}$, $a_R \leftarrow y_{(l)}$ ▷ no auxiliary solve

stream r once: form $\tilde{S}_L, \tilde{S}_R$ of Eq. (13), count $|R|$, gather M

if CERTIFY **and** $(\Psi_r(a_R) > k$ **or** $\Psi_r(a_L) < k)$ **then** ▷ exact, one $\mathcal{O}(n)$ pass

  double the failing side of the bracket and repeat; after four failures set $[a_L, a_R) \leftarrow [r_{\min}, r_{\max}]$

end if

sort M and run Algorithm 1 on M with A, B initialized from $\tilde{S}_L, \tilde{S}_R, |R|$

return $p_i = \sigma\big((r_i - b)/t\big)$ for all $i \in \{1, ..., n\}$

A single forward pass computes the tail sums, $|R|$, and gathers the middle set.

The bracketed solve then proceeds identically to Sec. B, but relies on two augmented coefficients. After sorting M in descending order as $s_1^M \ge \cdots \ge s_{|M|}^M$, we write $b = s_i^M + t\delta$ and set $m = e^\delta$.

Note that the R-tail term carries the factor $e^{-(a_R - s_i^M)/t} m$, meaning it naturally joins the m-family of the"above" group, while the L-tail term carries $e^{-(s_i^M - a_L)/t} m^{-1}$ and aligns with the "below" group. Hence, by defining

$$A_i^{\mathrm{aug}} = A_i + \tilde{S}_R\, e^{-(a_R - s_i^M)/t}, \qquad B_i^{\mathrm{aug}} = B_i + \tilde{S}_L\, e^{-(s_i^M - a_L)/t}, \qquad n_i^{\mathrm{aug}} = |R| + i - 1, \tag{15}$$

the node values become $\Psi_r(s_i^M) = n_i^{\mathrm{aug}} + \frac{1}{2}(1 + B_i^{\mathrm{aug}} - A_i^{\mathrm{aug}})$, and the interval equation takes the quadratic form:

$$A_{i^\star}^{\mathrm{aug}} m^2 - 2\big(n_{i^\star}^{\mathrm{aug}} - k\big)m - \big(1 + B_{i^\star}^{\mathrm{aug}}\big) = 0, \tag{16}$$

which is solved exactly as in Eq. (11) using these augmented coefficients. Since $a_L \le s_i^M \le a_R$, all newly introduced exponents remain non-positive, preserving the numerical stability of the scan. As long as $b \in [a_L, a_R)$, this bracketed formulation evaluates to the exact same threshold as Algorithm 1 up to standard floating-point precision, fulfilling the (P1) property discussed in Sec. 4.3. Furthermore, boundary cases where b lies above all elements of M, below all of them, or M is empty reduce to the same equation with A^{aug} or B^{aug} containing only the tail term.

Algorithm 2 summarizes the complete sample-bracketed Fast LapSum procedure. The probabilistic construction of the bracket and its coverage guarantees are developed in Appendix Section D.

D THE PROBABILISTIC BRACKET: GUARANTEES, SAMPLER, AND RUNTIME

THE THRESHOLD IS A KERNEL-NOISED QUANTILE

For $\alpha < 0$ the Laplace CDF gives $\sigma((r_i - b)/t) = \Pr(r_i + \varepsilon \geq b)$ with $\varepsilon \sim \text{Laplace}(0, t)$, so

$$\Psi_r(b) = \sum_i \sigma\left(\frac{r_i - b}{t}\right) = n \Pr(Y \geq b), \qquad Y = r_I + \varepsilon, \quad I \sim \text{Unif}\{1, \ldots, n\}, \tag{17}$$

where I and ε are independent random variables. Setting $\Psi_r(b) = k$ gives $\Pr(Y < b) = (n - k)/n =: q$, i.e. the threshold b is exactly the q-quantile of the kernel-noised mixture Y.

A BINOMIAL BRACKET FROM TWO ORDER STATISTICS

Draw K indices uniformly with replacement and form the kernel-noised sample $y_s = r_{i_s} + \varepsilon_s$ (ε_s i.i.d.), sorted ascending $y_{(1)} \leq \cdots \leq y_{(K)}$. By Eq. (17) each draw satisfies $y_s \leq b$ independently with probability q, so $C = \#\{s : y_s \leq b\}$ is exactly binomial and the order statistics bracket b.

Theorem 1. *Let $C \sim \text{Binomial}(K, q)$ with $q = (n - k)/n$. For integers $1 \leq j < l \leq K$,*

$$\Pr\left(y_{(j)} \leq b < y_{(l)}\right) = \Pr\left(j \leq C \leq l - 1\right). \tag{18}$$

Proof. The indicators $\mathbf{1}[y_s \leq b]$ are i.i.d. $\text{Bernoulli}(q)$ by Eq. (17), so $C \sim \text{Binomial}(K, q)$. Now $y_{(j)} \leq b$ iff at least j draws fall at or below b, i.e. $C \geq j$, and $b < y_{(l)}$ iff fewer than l do, i.e. $C \leq l - 1$. Intersecting the two events gives the claim. □

Choosing the indices. While Theorem 1 establishes an exact finite-sample identity for coverage, it leaves the choice of the indices $j < l$ open. We consider three distinct strategies for determining these bounds:

$$j = \left\lfloor Kq - z\sqrt{Kq(1-q)} \right\rfloor, \qquad l = \left\lceil Kq + z\sqrt{Kq(1-q)} \right\rceil. \tag{19}$$

For extreme budgets, the sample quantile indices may fall outside $[1, K]$. The implementation clamps the indices to the sample range and extends the affected endpoint by $10|\alpha|$. This case is outside the sampling-coverage result above. The endpoint certificate accepts a valid bracket, widens a missed bracket, and invokes the exact full-sort fallback after four failed attempts. Consequently, index clipping can affect runtime but not correctness.

- *Exact binomial quantiles.* Take $j = \max\{u : \Pr(C < u) \leq \eta/2\}$ and $l = \min\{u : \Pr(C \geq u) \leq \eta/2\}$, evaluated once from the regularized incomplete beta function. Theorem 1 then gives coverage $\geq 1 - \eta$ with no approximation. The cost is a single scalar special-function evaluation per call, which is negligible. This is the setting we recommend when a stated finite-sample guarantee is required.
- *Bernstein bound.* With j, l as in Eq. (19), the failure event is $|C - Kq| > z\sqrt{Kq(1-q)}$, and Bernstein's inequality for bounded summands gives the distribution-free bound

$$\Pr\left(b \notin [a_L, a_R)\right) \;\leq\; 2\exp\left(-\frac{z^2}{2\left(1 + \frac{z}{3\sqrt{Kq(1-q)}}\right)}\right) \;=\; 2\exp\Big(-\tfrac{1}{2}z^2\big(1 - o(1)\big)\Big), \tag{20}$$

  where the $o(1)$ is explicit and vanishes as $K \to \infty$. We prefer Bernstein's inequality over Hoeffding's because the target quantile $q = (n - k)/n$ is typically far from $1/2$. In this highly skewed regime, the Bernoulli variance $q(1 - q)$ is small, rendering Hoeffding's variance-agnostic bound of $2\exp(-2z^2q(1 - q))$ overly loose.
- *Normal approximation.* Eq. (19) with the reading "coverage $\geq 1 - 2\Phi(-z)$" is a normal approximation, not a bound. By the Berry–Esseen theorem the standardized binomial CDF is within $\mathcal{O}(K^{-1/2}) = \mathcal{O}(n^{-1/3})$ of the normal one, so the correct statement is coverage $= 1 - 2\Phi(-z) + \mathcal{O}(n^{-1/3})$. While this normal approximation is often colloquially cited as a strict guarantee, the finite-sample error term necessitates relying on the exact or Bernstein bounds for formal claims.

For our default margin $z = 5.16$, the Bernstein bound in Eq. (20) restricts the theoretical failure probability to strictly below 10^{-5}. This ensures **the initial bracket successfully captures b in nearly all instances under i.i.d. sampling, avoiding the need for subsequent bracket widening**. Importantly, because this localization relies entirely on two empirical order statistics, it introduces no auxiliary LapSum overhead. Finally, balancing the $\mathcal{O}(K \log K)$ sorting cost of the sample against the sorting of the localized set M directly motivates our sublinear sample size choice of $K \approx n^{2/3}$; a coarser $K \approx 2\sqrt{n}$ sample is also available when a wider bracket is acceptable.

D.1 From the Analyzed Sampler to the Deployed One

Theorem 1 assumes K i.i.d. draws with replacement. The deployed fast path does not satisfy this assumption, so we analyze the sampling procedures separately.

With replacement (analyzed). The coverage theorem applies verbatim. The runtime result in Prop. 3 additionally draws a fresh i.i.d. sample on every retry. This is the variant to which the theoretical failure probabilities refer.

Without replacement. Drawing K distinct indices uniformly makes C hypergeometric. The identity of Theorem 1 still holds with this distribution. By the reduction of Hoeffding (1963), every convex tail bound valid for the binomial case remains valid, so Eq. (20) is unchanged. Sampling without replacement is at least as concentrated under these bounds.

Systematic (deployed). For throughput the fast path takes a strided view of the score buffer, $i_s = (o + s\lceil n/K \rceil) \bmod n$, and adds pre-computed kernel noise, which avoids a gather and a per-call random-number pass. This constitutes systematic sampling, which naturally lacks the independence or exchangeability assumptions required by the theoretical bounds above. The primary failure case arises if the scores are arranged periodically and the period shares a large common factor with the stride, the sample is drawn from a subpopulation whose quantiles differ from those of the full vector, and the bracket can miss b by an arbitrary margin. Sorted or block-structured inputs are realistic instances. We randomize the offset, keep $z = 5.16$, and measure coverage on the ten score distributions in Sec. 6.3. These choices do not constitute a proof. Correctness comes from the certification pass of Algorithm 2, which directly evaluates both bracket endpoints and widens any missed bracket.

D.2 Relation to Probabilistic Hard Selection (Prof-K)

Sampled thresholds combined with a one-pass filter are a standard technique in selection algorithms, most recently instantiated for hard top-k by the concurrent Prof-K (Dziarmaga et al., 2026). While the high-level structures appear similar, a budget-exact soft operator introduces fundamental differences. We highlight four key distinctions that necessitated the design of Algorithm 2.

What is being localized. Prof-K localizes an order statistic of the scores, the k-th largest value. Our threshold is not an order statistic. By Eq. (17), it is the $(n-k)/n$ quantile of the kernel-noised mixture $Y = r_I + \varepsilon$. The sample must therefore be noised before sorting, and the binomial bracket uses $q = (n-k)/n$. At temperature $t \to 0$, the two thresholds coincide.

What happens to the discarded elements. A hard selector may drop everything outside the buffer because those elements cannot enter the answer. A budget-exact soft operator must retain their aggregate contribution because every score contributes mass to $\sum_i p_i = k$. The two exact sums in Eq. (13) summarize the tails, which re-enter the interval equation through the augmented coefficients in Eq. (15). This step makes the bracketed solve identical to the full solve.

Form of the guarantee. Prof-K returns the correct hard top-k with probability at least $1 - \epsilon$ for a user-specified ϵ, distribution-free. Our guarantees are split differently (Sec. 4.3): the budget is exact deterministically given a valid bracket, the bracket is probabilistic only in the uncertified fast path, and certification makes it deterministic at the cost of one $\mathcal{O}(n)$ pass, since $\Psi_r(a_L), \Psi_r(a_R)$ can be evaluated exactly, which has no analogue for a hard selector, where verifying a candidate set is as expensive as producing it. Conversely, Prof-K's distribution-free buffer-size analysis is an alternative to the index choice Eq. (19) when a stated ϵ is preferred to certification, and the two are compatible: any bracket rule can be plugged into Algorithm 2 without touching the solve.

Consequences for cost. Prof-K reports $1.5\times$–$10\times$ speedups over optimized hard top-k implementations at large n with small to moderate k, i.e. the regime where our bracketed branch also pays off. Read together with Fig. 2, this supports the claim of Sec. 5 that the differentiable operator is within a constant factor of a probabilistic hard selector, and that the constant is not large.

Empirical Coverage

The theoretical coverage result above applies to i.i.d. sampling with replacement. We separately test that sampler, sampling without replacement, and the deployed systematic sampler in Sec. 6.3. The calibrated coverage curves and the $n = 10^7$ large-scale evaluation are reported in Fig. 4 and Table 5.

Expected Linear Runtime

In this section, we formally analyze the expected $\mathcal{O}(n)$ time complexity of our bracketed solver under an i.i.d. sampling scheme. Because the running time is dominated by the size of the middle set M, we must bound the empirical score concentrations using the kernel-noised mixture Y. Lemma 1 provides this critical mathematical link, motivating hypotheses required by Prop. 3 to prove an expected linear runtime. Although our deployed systematic sampler substitutes i.i.d. sampling for hardware-optimized reads and is therefore evaluated empirically, this analysis justifies the $\mathcal{O}(n)$ runtime.

Lemma 1 (Kernel mass transfer)**.** *Let $Y = r_I + \varepsilon$ with $\varepsilon \sim \mathrm{Laplace}(0, t)$ as in Eq. (17). For any interval $\mathcal{I} = [a, a + w)$ with $w > 0$,*

$$\#\{i : r_i \in \mathcal{I}\} \;\le\; C(w/t)\, n \,\Pr(Y \in \mathcal{I}), \qquad C(u) = \frac{2}{u} e^{u}, \tag{21}$$

and C attains its minimum $2e$ at $u = 1$.

Proof. Let $m = \#\{i : r_i \in \mathcal{I}\}$. The density of Y is $f_Y(y) = \frac{1}{n}\sum_i \frac{1}{2t} e^{-|y-r_i|/t}$. For $y \in \mathcal{I}$ and $r_i \in \mathcal{I}$ we have $|y - r_i| < w$, so $f_Y(y) \ge \frac{m}{2tn} e^{-w/t}$. Integrating over $\mathcal{I}$ gives $\Pr(Y \in \mathcal{I}) \ge \frac{mw}{2tn} e^{-w/t}$, which rearranges to the claim. The stated minimum follows from $C'(u) = 0$ at $u = 1$. □

Remark 1. Lemma 1 shows that raw scores cannot hide from the noised mixture at scales comparable to the kernel width: for $w \asymp t$ the middle set is at most a constant multiple of $n \Pr(Y \in \mathcal{I})$. It does *not* rule out concentration at scales far below t, where the factor $C(w/t) \approx 2t/w$ degrades: if a constant fraction of the scores is tied at a single value inside a bracket of width $w \ll t$, then $|M| = \Theta(n)$ and the bound is tight. This is exactly the configuration that breaks linear time, and it is why Prop. 3 assumes (H3) and why the fallback of Algorithm 2 matters.

Proposition 3 (Expected linear time of the i.i.d. retry variant)**.** *Consider the following analyzed variant of Algorithm 2. Set $K = \lceil n^{2/3} \rceil$. On attempt $\tau = 0, 1, \ldots$, draw a fresh i.i.d. sample of K kernel-noised scores, set $z_\tau = 2^\tau z_0$, and compute*

$$j_\tau = \left\lfloor Kq - z_\tau \sqrt{Kq(1-q)} \right\rfloor, \qquad l_\tau = \left\lceil Kq + z_\tau \sqrt{Kq(1-q)} \right\rceil. \tag{22}$$

The bracket is accepted only after certification. If either index ceases to be interior, the algorithm invokes the exact full-sort solver. Assume

- *(H1) every attempt uses a fresh i.i.d. sample from Y;*
- *(H2) the density f_Y is bounded below by $c_1 > 0$ on a neighbourhood $\mathcal{N}$ containing the brackets considered before fallback;*
- *(H3) the empirical score distribution is regular on $\mathcal{N}$: there is $c_3 < \infty$ with $\#\{i : r_i \in \mathcal{I}\} \le c_3(n|\mathcal{I}| + 1)$ for every interval $\mathcal{I} \subseteq \mathcal{N}$;*
- *(H4) the target quantile is bounded away from both extremes: there exists a constant $\delta > 0$ such that $\delta \le q \le 1 - \delta$.*

Then the expected running time is $\mathbb{E}[T] = \mathcal{O}(n)$, and the expected total number of middle-set elements processed across all attempts is $\mathcal{O}(n^{2/3})$. Without these assumptions, certification still preserves correctness and the full-sort fallback gives $\mathcal{O}(n \log n)$ worst-case complexity.

Proof. Let C_τ be the number of samples on attempt τ that do not exceed the exact threshold. By (H1), $C_\tau \sim \mathrm{Binomial}(K, q)$. The bracket misses the threshold only if $C_\tau < j_\tau$ or $C_\tau \geq l_\tau$. By Eq. (20), while the indices remain interior there is a constant $c_{\mathrm{B}} > 0$ such that

$$\rho_\tau := \Pr\big(b \notin [a_{L,\tau}, a_{R,\tau})\big) \leq 2\exp(-c_{\mathrm{B}} z_\tau^2) = 2\exp(-c_{\mathrm{B}} 4^\tau z_0^2). \tag{23}$$

This bound is obtained directly in binomial rank space and makes no claim that doubling a score interval exactly doubles a standardized rank margin.

The expected probability mass between the two order statistics on attempt τ is

$$\mathbb{E}[F_Y(a_{R,\tau}) - F_Y(a_{L,\tau})] = \frac{l_\tau - j_\tau}{K+1} = \mathcal{O}\left(\frac{z_\tau}{\sqrt{K}}\right). \tag{24}$$

By (H2), the expected bracket width is therefore $\mathcal{O}(z_\tau/(c_1\sqrt{K}))$. Assumption (H3) then gives

$$\mathbb{E}|M_\tau| = \mathcal{O}\left(\frac{c_3}{c_1}\frac{z_\tau n}{\sqrt{K}} + 1\right) = \mathcal{O}(2^\tau n^{2/3}). \tag{25}$$

The density and score-regularity assumptions enter only this middle-set bound; the failure probability in Eq. (23) is a binomial rank-space statement.

Fresh samples make the conditional failure bound on each reached attempt equal to Eq. (23). Consequently,

$$\Pr(\text{reach attempt } \tau) \leq \prod_{s=0}^{\tau-1} 2\exp(-c_{\mathrm{B}} 4^s z_0^2) = 2^\tau \exp\left(-c_{\mathrm{B}} z_0^2 \frac{4^\tau - 1}{3}\right). \tag{26}$$

One attempt requires an $\mathcal{O}(n)$ stream, which inherently includes the deterministic certification check $\Psi_r(a_R) \leq k \leq \Psi_r(a_L)$, an $\mathcal{O}(K \log K)$ sample sort, and an $\mathcal{O}(|M_\tau| \log n)$ middle-set sort. Combining Eqs. (25) and (26),

$$\mathbb{E}[T] = \mathcal{O}\left(n \sum_{\tau \geq 0} \Pr(\text{reach } \tau) + n^{2/3} \log n \sum_{\tau \geq 0} 2^\tau \Pr(\text{reach } \tau)\right) = \mathcal{O}(n). \tag{27}$$

Both series converge because the factor in Eq. (26) decays doubly exponentially. The same calculation without the logarithmic sorting factor gives an expected total middle-set size of $\mathcal{O}(n^{2/3})$.

By (H4), an index in Eq. (22) first ceases to be interior only when $z_\tau^2 = \Theta(K)$. equation 26 bounds the probability of reaching this point by $\exp[-\Omega(K)]$. The expected contribution of the $\mathcal{O}(n \log n)$ full-sort fallback is therefore $\mathcal{O}(n \log n\, e^{-\Omega(K)}) = o(n)$. □

While this expected linear-time guarantee requires the i.i.d. hypothesis (H1), the exact certification ensures that substituting a faster systematic sampler in deployment only impacts the empirical fallback rate, never the exactness of the final output.

Scope of the result. The proposition applies to the analyzed i.i.d. retry-until-certified variant. The deployed implementation uses systematic sampling and a finite retry cap, and is not covered by this expected-linear theorem. Its scaling is established empirically. In both variants, certification rejects a missed bracket, and the exact full-sort fallback preserves correctness with $\mathcal{O}(n \log n)$ worst-case complexity. Negligible density at the threshold violates (H2), heavy ties can violate (H3), and extreme budgets violate (H4). These cases affect the runtime statement, not the validity of an accepted output. On the initial i.i.d. attempt, Eq. (25) gives $\mathbb{E}|M_0| = \mathcal{O}(n^{2/3})$; the implementation caps the gather buffer and falls back when this cap is exceeded.

E Implementation Details

All forward implementations use the same autograd interface. The reference solver performs 60 vectorized bisection steps. The PyTorch implementation replaces bisection with the full-sort scan in Algorithm 1. The CUDA implementation evaluates the affine scan cooperatively, with one block assigned to each batch element. Its fused version computes the final CDF in the same kernel. All variants use the analytical backward pass in Eq. (5).

Variant	Forward (ms)	Speedup
Bisection (60 steps)	7.614	1.0×
PyTorch full scan	0.647	11.8×
Cooperative CUDA scan	0.206	36.9×
Fused CUDA scan and CDF	**0.105**	72.6×

Table 6: Median forward runtime on an NVIDIA GH200 120GB ($B = 16$, $n = 4096$, $k = 256$, $\alpha = -1$). Speedup is measured against 60-step bisection. Bold marks the lowest runtime and highest speedup.

Replacing bisection with the PyTorch full scan reduces the runtime from 7.614 to 0.647 ms. The cooperative CUDA scan and CDF fusion reduce it further to 0.105 ms, a $72.6\times$ speedup over bisection.

The bracketed implementation first samples and perturbs scores with Laplace noise. Sorting this sample gives the endpoints $a_L = y_{(j)}$ and $a_R = y_{(l)}$. One pass over the input computes the two tail sums, evaluates the budget curve at both endpoints, and gathers the middle set M. Only M is sorted, after which the exact scan is applied with the augmented coefficients in Eq. (15).

Table 7: Analytical VJP error against centered finite differences from an independent 200-step FP64 bisection reference. The relative-error tolerance is 10^{-3}.

B	n	k	α	Abs. error	Rel. error
2	32	4	-1.00	3.15e-11	8.01e-11
1	64	8	-0.50	1.19e-10	7.37e-11
2	64	16	-2.00	9.67e-11	2.30e-10

Certification. The endpoint test accepts a bracket when $\Psi_r(a_R) \le k \le \Psi_r(a_L)$, using an FP32 tolerance of $10^{-4}\max(k, 1)$. If the test fails, the margin z is doubled and the bracket is recomputed. After four failed attempts, the exact full-sort solver is used. The operator comparisons in Sec. 5, Figure 7, and Tables 8 and 9 measure the complete certified path, including retries and fallback when triggered. Initial bracket coverage and retry rates are reported separately in Sec. 6.3.

Numerical precision. The experiments use FP32. FP64 is used only for the finite-difference reference in Tab. 7. For the interval root, we select between the two forms in Eq. (11) according to the sign of $i^\star - 1 - k$. This avoids subtractive cancellation and remains well-defined as $A_{i^\star} \to 0$.

F Additional Speed and Ablation Results

Fig. 7 includes the analytical vector–Jacobian product in Eq. (5). At $n = 3\times10^4$, the certified median forward and backward time is 1.344 ms for Fast LapSum, 40.952 ms for NeuralSort, and 31.839 ms for SoftSort.

F.1 Comparison with DFTopK

DFTopK (Zhu et al., 2025) also studies linear-time differentiable top-k. Its application experiments use RecFlow and an internal advertising system, so we restrict the comparison to the operator benchmark from its Table 2.

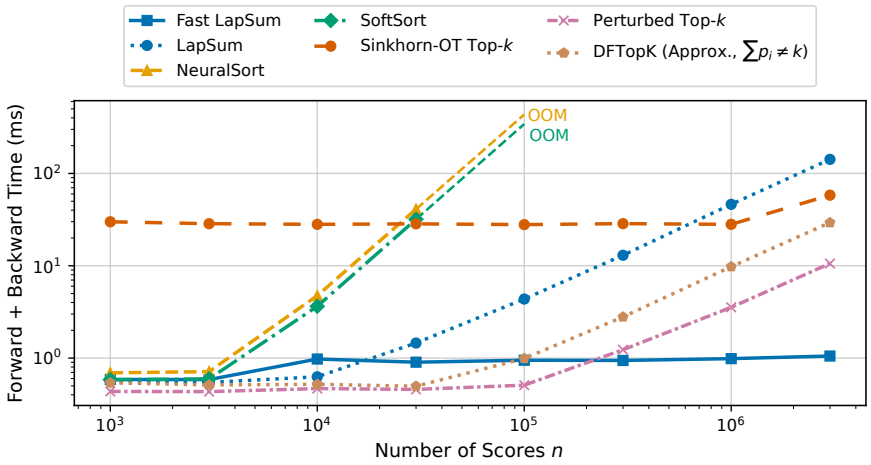

Figure 7: Median forward and backward runtime on an NVIDIA GH200 120GB (B=1, k=$n/16$, FP32). The Fast LapSum curve includes endpoint certification and any retries.

N	LapSum (ms)	Certified Fast LapSum (ms)	Certified p95 (ms)	Retry (%)	Fallback (%)	Speedup over LapSum
1×10^6	41.463	**0.923**	0.972	0.0	0.0	44.9×
3×10^6	127.739	**1.008**	1.052	0.0	0.0	126.7×
1×10^7	432.363	**1.390**	2.291	6.7	0.0	311.1×
3×10^7	1409.621	**2.818**	6.027	29.2	0.0	500.2×
1×10^8	5240.809	**7.241**	15.576	33.3	0.0	723.8×
3×10^8	16468.881	**19.498**	65.935	45.0	0.0	844.6×
1×10^9	55591.469	**121.764**	232.446	60.0	0.0	456.5×

Table 8: Forward runtime on an NVIDIA GH200 120GB (B=1, k=$N/16$, α= $-$ 1). Fast LapSum timings include endpoint certification and retries. Retry is the percentage of timed calls requiring more than one sampled bracket. Fallback is the percentage reaching the exact full-sort path. Speedup is the LapSum median divided by the certified Fast LapSum median. The $456.5\times$ value at $N = 10^9$ is reported as $457\times$ in the main text. Bold marks the lower median runtime.

Protocol. We measure one forward and backward pass with $k = \lfloor n/2 \rfloor$ and batch size 1. All methods run on the same NVIDIA GH200 120GB in FP32. The comparison includes DFTopK, the official LapSum implementation (Struski et al., 2025), Fast LapSum, and the differentiable-ordering baselines from Fig. 2. The original DFTopK table stops at $n = 10^3$. We report sizes up to $n = 3{\times}10^6$.

Table 9: Median forward and backward runtime in milliseconds on an NVIDIA GH200 120GB ($k = \lfloor n/2 \rfloor$, batch size 1, FP32). The Fast LapSum measurements include endpoint certification and retries. "Mem." indicates that the estimated quadratic live tensors exceed 85% of device memory. Bold marks the lowest runtime in each row.

n	DFTopK	Fast LapSum	LapSum	NeuralSort	SoftSort	Sinkhorn-OT	Perturbed
10^3	0.510	0.626	0.560	0.662	0.557	29.830	**0.420**
10^4	0.476	1.586	0.634	4.906	3.796	29.202	**0.454**
10^5	0.795	1.651	4.374	Mem.	Mem.	28.938	**0.556**
10^6	6.554	**2.555**	46.579	Mem.	Mem.	28.686	3.877
$3{\times}10^6$	23.087	**4.345**	144.005	Mem.	Mem.	58.561	11.528

Operator \ Size	10^3	10^4	10^5	10^6	10^7
Fast LapSum (ours)	6.79e−8	8.17e−8	1.45e−7	2.62e−8	6.52e−8
LapSum (Struski et al., 2025)	1.13e−7	2.34e−8	1.46e−6	1.12e−5	1.54e−3
DFTopK (Zhu et al., 2025)	2.39	2.44	2.46	2.46	2.46
NeuralSort (Grover et al., 2019)	1.99e−9	2.04e−9	Mem.	Mem.	Mem.
SoftSort (Prillo & Eisenschlos, 2020)	1.30e−9	2.71e−10	Mem.	Mem.	Mem.
Sinkhorn-OT (Xie et al., 2020)	2.41e−8	2.11e−8	6.92e−8	3.26e−8	4.40e−8
Perturbed (Berthet et al., 2020)	2.68e−8	2.67e−8	2.57e−8	2.58e−8	2.58e−8

Table 10: Relative budget error $|\sum_i p_i - k|/k$ on an NVIDIA GH200 120GB ($k{=}N/16$, $\alpha{=}-1$, Gaussian scores). Fast LapSum uses the complete certified path. "Mem." indicates that the estimated quadratic live tensors exceed 85% of device memory. DFTopK does not impose $\sum_i p_i = k$. Bold marks the lowest measured error in each column.

Results. Tab. 9 reports the measured runtimes. At $n = 10^3$, DFTopK, certified Fast LapSum, and LapSum take 0.510, 0.626, and 0.560 ms, respectively. At $n = 3{\times}10^6$, their runtimes are 23.087, 4.345, and 144.005 ms. Fast LapSum is therefore $5.3\times$ faster than DFTopK and $33.1\times$ faster than LapSum at the largest common size. NeuralSort and SoftSort exceed the memory criterion from $n = 10^5$ because they construct quadratic tensors. DFTopK does not impose the LapSum budget equation $\sum_i p_i = k$. The resulting budget error is reported in Tab. 10.

G Application Protocols

G.1 3D Gaussian Splatting

Standard scenes. For each scene and seed, the full ADC run gives the reference count N_{ADC}. We set $K = \mathrm{round}(0.3N_{\mathrm{ADC}})$. ADC runs through iteration $10{,}000$, after which densification is disabled. From iterations $10{,}000$ to $12{,}000$, the soft budget follows a cubic schedule from the active count N to K, and the temperature decreases from 3.0 to 0.1. The scores are optimized with Adam at learning rate 0.02. No Gaussians are physically removed during this interval. At iteration $12{,}000$, the mask is recomputed after the optimizer update, the highest-scoring K Gaussians are retained, and the selector is removed. The retained set is trained until iteration $30{,}000$.

The footprint baseline uses the same activation iteration, value of K, hard-pruning iteration, and recovery period. Its ranking is determined by the accumulated footprint statistic. The retention sweep uses bicycle, garden, kitchen, playroom, and train with seed 0. The final comparison uses all 13 scenes with seeds 0, 1, and 2.

During the soft stage, we store fixed-bin histograms of the mask probabilities on $[0, 1]$ and their quantiles at fixed checkpoints. A final snapshot is taken after the mask recomputation at iteration $12{,}000$ and before hard top-K pruning. The same logging is used before the large-scene transition at iteration $36{,}000$. Diagnostic logging is excluded from the reported training time.

Large scenes. The monolithic runs use the same 30% retention target and score optimizer. Densification ends at iteration $30{,}000$, score optimization ends with hard pruning at iteration $36{,}000$, and training ends at iteration $60{,}000$. These runs do not use spatial blocks or CPU offloading. The LapSum comparison changes only the threshold solver.

Measurements. PSNR, SSIM, and LPIPS are evaluated on held-out views. Training time is measured with GPU synchronization and excludes metric evaluation and diagnostic logging. Rendering throughput is measured from the saved model, and model size is the size of the final PLY file.

Per-scene results. Tabs. 11 and 12 give the complete results underlying the 13-scene averages in Tab. 1. The two pruning methods use the same per-scene Gaussian budget.

Qualitative comparisons. Figs. 8 to 12 compare ADC and Fast LapSum on all 13 scenes. For each scene, we select the ADC held-out view whose PSNR is closest to that scene's median per-view PSNR, then display the same view for both methods.

Table 11: Per-scene image quality at 30% retention. Results are means ± standard deviations over three seeds. Bold marks the best mean within each scene.

Scene	Method	PSNR (dB) ↑	SSIM ↑	LPIPS ↓
Bicycle	ADC	**25.27 ± 0.03**	**0.767 ± 0.000**	**0.208 ± 0.000**
	+ hard pruning	25.21 ± 0.04	0.757 ± 0.000	0.233 ± 0.001
	+ Fast LapSum	25.25 ± 0.06	0.761 ± 0.001	0.232 ± 0.001
Flowers	ADC	**21.58 ± 0.00**	**0.605 ± 0.001**	**0.336 ± 0.000**
	+ hard pruning	21.48 ± 0.01	0.596 ± 0.001	0.357 ± 0.001
	+ Fast LapSum	21.49 ± 0.04	0.597 ± 0.000	0.354 ± 0.000
Garden	ADC	**27.44 ± 0.04**	**0.867 ± 0.001**	**0.107 ± 0.000**
	+ hard pruning	26.90 ± 0.03	0.847 ± 0.001	0.139 ± 0.000
	+ Fast LapSum	27.41 ± 0.05	0.865 ± 0.001	0.116 ± 0.000
Stump	ADC	26.60 ± 0.01	0.771 ± 0.001	**0.216 ± 0.001**
	+ hard pruning	26.56 ± 0.05	0.770 ± 0.001	0.225 ± 0.001
	+ Fast LapSum	**26.63 ± 0.04**	**0.773 ± 0.001**	0.229 ± 0.001
Treehill	ADC	22.50 ± 0.07	**0.634 ± 0.001**	**0.324 ± 0.001**
	+ hard pruning	22.56 ± 0.07	0.630 ± 0.001	0.350 ± 0.001
	+ Fast LapSum	**22.67 ± 0.04**	0.634 ± 0.001	0.349 ± 0.000
Room	ADC	**31.57 ± 0.09**	**0.920 ± 0.000**	**0.217 ± 0.001**
	+ hard pruning	31.21 ± 0.16	0.912 ± 0.000	0.236 ± 0.001
	+ Fast LapSum	31.49 ± 0.10	0.919 ± 0.000	0.222 ± 0.001
Counter	ADC	**29.10 ± 0.00**	**0.909 ± 0.000**	**0.199 ± 0.000**
	+ hard pruning	28.63 ± 0.03	0.894 ± 0.000	0.228 ± 0.000
	+ Fast LapSum	28.96 ± 0.03	0.906 ± 0.000	0.207 ± 0.000
Kitchen	ADC	**31.44 ± 0.07**	**0.928 ± 0.000**	**0.125 ± 0.000**
	+ hard pruning	31.08 ± 0.04	0.921 ± 0.000	0.139 ± 0.000
	+ Fast LapSum	31.26 ± 0.24	0.927 ± 0.001	0.129 ± 0.000
Bonsai	ADC	**32.27 ± 0.11**	**0.942 ± 0.001**	**0.203 ± 0.000**
	+ hard pruning	31.77 ± 0.10	0.934 ± 0.001	0.220 ± 0.000
	+ Fast LapSum	31.97 ± 0.04	0.938 ± 0.000	0.211 ± 0.000
Train	ADC	22.20 ± 0.14	**0.823 ± 0.001**	**0.195 ± 0.000**
	+ hard pruning	21.93 ± 0.07	0.800 ± 0.001	0.240 ± 0.001
	+ Fast LapSum	**22.22 ± 0.07**	0.817 ± 0.001	0.216 ± 0.001
Truck	ADC	**25.42 ± 0.02**	**0.885 ± 0.000**	**0.142 ± 0.000**
	+ hard pruning	25.13 ± 0.07	0.878 ± 0.000	0.156 ± 0.000
	+ Fast LapSum	25.40 ± 0.01	0.884 ± 0.001	0.147 ± 0.001
Dr. Johnson	ADC	29.44 ± 0.06	0.905 ± 0.000	**0.236 ± 0.000**
	+ hard pruning	29.24 ± 0.08	0.898 ± 0.000	0.260 ± 0.000
	+ Fast LapSum	**29.49 ± 0.02**	**0.907 ± 0.000**	0.242 ± 0.001
Playroom	ADC	**30.20 ± 0.13**	0.909 ± 0.001	**0.240 ± 0.001**
	+ hard pruning	29.98 ± 0.05	0.906 ± 0.001	0.256 ± 0.000
	+ Fast LapSum	30.19 ± 0.06	**0.910 ± 0.000**	0.247 ± 0.000

Table 12: Per-scene representation size and runtime at 30% retention. Results are means ± standard deviations over three seeds. Bold marks the best mean within each scene. Equal values arise because the two pruning methods retain the same exact number of Gaussians.

Scene	Method	N (M) ↓	Train (min) ↓	FPS ↑	Model (MiB) ↓
Bicycle	ADC	4.88 ± 0.01	14.19 ± 0.14	116.3 ± 0.3	1155.3 ± 3.3
	+ hard pruning	$\mathbf{1.47 \pm 0.01}$	10.65 ± 0.04	182.4 ± 2.7	$\mathbf{348.2 \pm 1.3}$
	+ Fast LapSum	$\mathbf{1.47 \pm 0.01}$	$\mathbf{9.61 \pm 0.08}$	$\mathbf{317.6 \pm 2.2}$	$\mathbf{348.2 \pm 1.3}$
Flowers	ADC	2.89 ± 0.02	10.36 ± 0.10	234.7 ± 5.7	682.8 ± 4.3
	+ hard pruning	$\mathbf{0.87 \pm 0.00}$	8.34 ± 0.18	371.8 ± 1.7	$\mathbf{206.4 \pm 0.3}$
	+ Fast LapSum	$\mathbf{0.87 \pm 0.00}$	$\mathbf{8.15 \pm 0.04}$	$\mathbf{455.5 \pm 9.1}$	$\mathbf{206.4 \pm 0.3}$
Garden	ADC	4.15 ± 0.01	13.07 ± 0.03	162.8 ± 0.5	981.2 ± 1.3
	+ hard pruning	$\mathbf{1.24 \pm 0.00}$	10.09 ± 0.08	264.6 ± 1.5	$\mathbf{294.0 \pm 0.8}$
	+ Fast LapSum	$\mathbf{1.24 \pm 0.00}$	$\mathbf{9.49 \pm 0.01}$	$\mathbf{352.3 \pm 11.1}$	$\mathbf{294.0 \pm 0.8}$
Stump	ADC	4.30 ± 0.04	10.96 ± 0.07	194.3 ± 2.7	1015.8 ± 9.8
	+ hard pruning	$\mathbf{1.27 \pm 0.02}$	8.24 ± 0.04	314.7 ± 0.4	$\mathbf{301.4 \pm 4.9}$
	+ Fast LapSum	$\mathbf{1.27 \pm 0.02}$	$\mathbf{7.90 \pm 0.12}$	$\mathbf{433.7 \pm 11.6}$	$\mathbf{301.4 \pm 4.9}$
Treehill	ADC	3.28 ± 0.02	11.33 ± 0.07	200.0 ± 2.3	775.7 ± 5.0
	+ hard pruning	$\mathbf{0.98 \pm 0.01}$	8.90 ± 0.08	316.6 ± 1.9	$\mathbf{231.9 \pm 1.2}$
	+ Fast LapSum	$\mathbf{0.98 \pm 0.01}$	$\mathbf{8.60 \pm 0.12}$	$\mathbf{415.2 \pm 4.7}$	$\mathbf{231.9 \pm 1.2}$
Room	ADC	1.31 ± 0.00	14.92 ± 0.07	195.5 ± 0.6	309.8 ± 1.1
	+ hard pruning	$\mathbf{0.40 \pm 0.00}$	12.95 ± 0.14	262.8 ± 8.2	$\mathbf{93.5 \pm 0.4}$
	+ Fast LapSum	$\mathbf{0.40 \pm 0.00}$	$\mathbf{12.90 \pm 0.14}$	$\mathbf{318.7 \pm 1.4}$	$\mathbf{93.5 \pm 0.4}$
Counter	ADC	1.08 ± 0.00	13.93 ± 0.07	208.5 ± 1.8	255.0 ± 0.9
	+ hard pruning	$\mathbf{0.32 \pm 0.00}$	12.29 ± 0.10	282.6 ± 2.8	$\mathbf{76.3 \pm 0.7}$
	+ Fast LapSum	$\mathbf{0.32 \pm 0.00}$	$\mathbf{12.14 \pm 0.08}$	$\mathbf{337.5 \pm 4.3}$	$\mathbf{76.3 \pm 0.7}$
Kitchen	ADC	1.61 ± 0.01	16.33 ± 0.12	178.9 ± 1.5	380.4 ± 1.5
	+ hard pruning	$\mathbf{0.48 \pm 0.00}$	13.78 ± 0.04	264.5 ± 1.1	$\mathbf{113.9 \pm 0.7}$
	+ Fast LapSum	$\mathbf{0.48 \pm 0.00}$	$\mathbf{13.22 \pm 0.04}$	$\mathbf{334.7 \pm 3.4}$	$\mathbf{113.9 \pm 0.7}$
Bonsai	ADC	1.07 ± 0.00	11.43 ± 0.12	289.4 ± 5.4	254.2 ± 1.0
	+ hard pruning	$\mathbf{0.32 \pm 0.00}$	$\mathbf{10.39 \pm 0.04}$	353.6 ± 13.0	$\mathbf{76.0 \pm 0.4}$
	+ Fast LapSum	$\mathbf{0.32 \pm 0.00}$	10.39 ± 0.10	$\mathbf{412.4 \pm 9.0}$	$\mathbf{76.0 \pm 0.4}$
Train	ADC	1.09 ± 0.01	8.60 ± 0.04	223.0 ± 0.7	258.2 ± 1.9
	+ hard pruning	$\mathbf{0.33 \pm 0.00}$	$\mathbf{6.64 \pm 0.02}$	413.2 ± 6.2	$\mathbf{77.3 \pm 0.7}$
	+ Fast LapSum	$\mathbf{0.33 \pm 0.00}$	6.64 ± 0.06	$\mathbf{455.6 \pm 25.1}$	$\mathbf{77.3 \pm 0.7}$
Truck	ADC	2.04 ± 0.01	8.65 ± 0.03	233.3 ± 3.2	482.9 ± 2.7
	+ hard pruning	$\mathbf{0.61 \pm 0.00}$	6.98 ± 0.02	358.7 ± 13.8	$\mathbf{144.9 \pm 1.1}$
	+ Fast LapSum	$\mathbf{0.61 \pm 0.00}$	$\mathbf{6.72 \pm 0.04}$	$\mathbf{448.8 \pm 21.6}$	$\mathbf{144.9 \pm 1.1}$
Dr. Johnson	ADC	3.16 ± 0.03	16.72 ± 0.05	155.8 ± 2.4	746.4 ± 8.3
	+ hard pruning	$\mathbf{0.95 \pm 0.01}$	13.33 ± 0.12	270.1 ± 0.7	$\mathbf{224.2 \pm 2.6}$
	+ Fast LapSum	$\mathbf{0.95 \pm 0.01}$	$\mathbf{13.00 \pm 0.15}$	$\mathbf{313.0 \pm 4.5}$	$\mathbf{224.2 \pm 2.6}$
Playroom	ADC	1.86 ± 0.01	13.21 ± 0.09	232.5 ± 2.0	439.5 ± 3.3
	+ hard pruning	$\mathbf{0.56 \pm 0.00}$	10.92 ± 0.03	377.7 ± 8.9	$\mathbf{131.8 \pm 1.1}$
	+ Fast LapSum	$\mathbf{0.56 \pm 0.00}$	$\mathbf{10.86 \pm 0.09}$	$\mathbf{422.2 \pm 5.8}$	$\mathbf{131.8 \pm 1.1}$



Figure 8: Qualitative comparison on Bicycle, Flowers, and Garden. Fast LapSum retains 30% of the ADC Gaussians.



Figure 9: Qualitative comparison on Bonsai, Counter, and Kitchen. Fast LapSum retains 30% of the ADC Gaussians.



Figure 10: Qualitative comparison on Room, Stump, and Treehill. Fast LapSum retains 30% of the ADC Gaussians.



Figure 11: Qualitative comparison on Train, Truck, and Dr. Johnson. Fast LapSum retains 30% of the ADC Gaussians.



Figure 12: Qualitative comparison on Playroom. Fast LapSum retains 30% of the ADC Gaussians.

G.2 Exact-Budget Adversarial Attacks

For an image $x \in [0,1]^{3\times H\times W}$, the attack optimizes one score per pixel, $r \in \mathbb{R}^{HW}$, and replacement-color logits z. Let p denote the soft mask and h the hard top-K mask:

$$h = \mathbf{1}[r \in \text{top-}K], \qquad \widehat{h} = p + \text{sg}(h - p), \qquad p = \text{LapSum}_K(r;\alpha), \tag{28}$$

and the attacked image is

$$x' = x \odot (1 - \widehat{h}) + \sigma(z) \odot \widehat{h}. \tag{29}$$

The forward pass uses $\widehat{h} = h$ and therefore selects exactly K pixel locations. The backward pass uses the derivative of the soft mask p. If an optimized replacement color equals the original color, the measured L_0 can be smaller than K. We minimize the untargeted Carlini–Wagner margin

$$\mathcal{L}_{\text{CW}} = f_y(x') - \max_{c\neq y} f_c(x'). \tag{30}$$

A negative margin indicates misclassification. We initialize r with Gaussian noise of standard deviation 10^{-3} and initialize z from the input with Gaussian noise of standard deviation 0.05. Both variables are optimized with Adam. The temperature decreases geometrically from 1.0 to 0.05, and the color learning rate is 0.05. On a separate set of 100 images, we selected score learning rates of 0.01 for ConvNeXt-V2 Tiny and 0.05 for ResNet-50. Each evaluation uses at most 500 steps and three restarts. Success is checked after quantizing the adversarial image to 8-bit color.

The ImageNet evaluation contains 1,000 validation images classified correctly by both frozen models. We use $K \in \{1, 5, 10, 50, 100, 500\}$. Sparse-PGD and PGD-$L_0$ use the same images, models, budgets, and 500-step limit. For DIV2K, we sort the 900 training and validation images by filename and select the first 15 for which ConvNeXt-V2 assigns at least 0.8 probability to its top-1 prediction. The first five are used for protocol development, and the remaining ten form the test set. The clean top-1 prediction is used as a pseudolabel. We evaluate $K \in \{100, 500, 1000, 1500, 2000\}$.