PROF-K: PROBABILISTIC ONE-PASS FILTERING FOR EFFICIENT TOP-k SELECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Top-k selection is a fundamental computational primitive with applications spanning databases, information retrieval, signal processing, and modern machine learning workloads, including sparse activations and attention pruning. As data sizes grow, existing approaches become inefficient: exact methods incur high memory and compute overhead, while approximate methods often rely on brittle heuristics that degrade under adversarial or heavy-tailed inputs. In this paper, we introduce Prof-K, a fast, scalable, and distribution-agnostic top-k algorithm with probabilistic correctness guarantees. Prof-K performs a single-pass filtering procedure: a small random sample estimates an adaptive threshold, the N input elements are streamed once into a compact buffer, and an exact top-k routine on this buffer recovers the true top-k elements with probability at least $1 - \varepsilon$, where $\varepsilon > 0$ is user specified. We derive high-probability guarantees for correctness and buffer size, together with an approximately optimal sample size that minimizes overhead as a function of N and k. Empirically, Prof-K achieves $1.5\times$–$15\times$ speedups over the highly optimized PyTorch `topk` and recent RadiK implementations, with the largest gains in the large-scale, small-to-moderate-k regime where prior methods struggle most. Unlike previous approaches, these guarantees hold independently of the input distribution, ensuring robustness to adversarial settings. A run-time check detects the rare failures, so the returned set can always be exact and ε bounds only the frequency of a second pass. We further demonstrate its impact on training BatchTopK Sparse Autoencoders (SAEs), where top-k selection constitutes a significant portion of the training cost. The source code is provided in the supplementary materials and will be made publicly available.

1 INTRODUCTION

Top-k selection is a core primitive of large-scale machine learning. It arises whenever only a small subset of the largest elements must be retained from a much larger collection (Akbarinia et al., 2011; Shanbhag et al., 2018), for example in sparse autoencoders (Bussmann et al., 2024) and gradient sparsification (Lin et al., 2018). As tensor sizes continue to grow, the cost of repeated top-k execution becomes increasingly significant in large-scale GPU workloads, particularly in sparse computation settings where only a small fraction of values is ultimately retained (Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022; Bussmann et al., 2024).

The challenge is most visible in the large-N, small-k regime. Even when only a tiny fraction of elements is needed, exact algorithms must still determine the precise global decision boundary across the full tensor. On modern GPUs, this cost is dominated primarily by memory movement and synchronization, making top-k disproportionately expensive relative to the surrounding computation (Alabi et al., 2012; Gaihre et al., 2021; Zhang et al., 2023; Xie et al., 2025; Li et al., 2024). Radix-based kernels such as RadiK (Li et al., 2024) reduce the latency but require auxiliary memory proportional to the input, and cannot run at all once the input approaches the device memory. In training workloads, where the selected tensor shares device memory with weights, activations and optimizer state, this overhead directly limits the admissible model or batch size.

In this work, we introduce Prof-K, a probabilistic one-pass filtering algorithm for exact top-k selection on GPUs. The method draws a small uniform sample to estimate a conservative threshold, streams once through the input while retaining only the elements above this threshold in a compact candidate

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

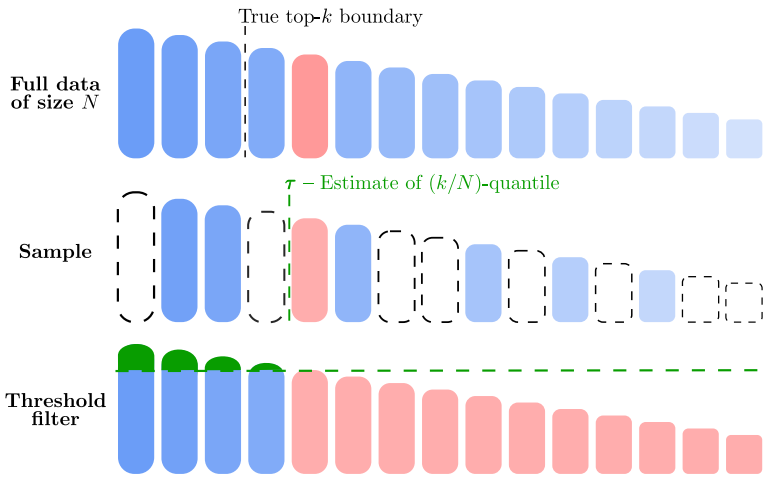


Figure 1: **The core idea of our Prof-K algorithm.** *Top:* Full data of size N sorted in descending order; the true top-k boundary is marked by a vertical dashed line. *Middle:* A random sample (solid outlines) is used to compute a threshold τ (green dashed line) as a conservative estimate of the (k/N)-quantile. *Bottom:* The threshold filter is applied in parallel to the full data. Elements $\geq \tau$ (green) are retained in a compact candidate buffer, while elements below τ are discarded. This approach ensures all true top-k elements are preserved with high probability while drastically reducing the workload for the final selection stage.

buffer, and applies an exact top-k routine to the candidates (Figure 1). The filter kernel counts the retained elements exactly, so the rare case in which the buffer is inadequate is detected at run time and handled by redrawing the sample: the output is always the exact top-k, and the user-specified budget ε bounds the frequency of the additional pass rather than the accuracy of the result. This distinguishes Prof-K from approximate top-k methods, which trade accuracy for speed (Chern et al., 2022). Because the analysis depends only on the ranks induced by uniform sampling, the guarantee holds for every input, including heavy-tailed and adversarial ones.

We summarize our contributions as follows.

- We propose Prof-K, a sampling-based filtering algorithm for exact top-k selection over the large tensors that arise in sparse ML workloads. It always returns the exact result: failures are detected at run time and resolved by resampling, with the budget ε bounding only their frequency, and the guarantee holds for every input distribution. We derive the optimal parameters for a prescribed failure budget, together with closed-form approximations that are accurate and cheap to evaluate.
- We evaluate Prof-K, implemented in Triton, against `torch.topk` and RadiK, the state-of-the-art radix selection, on a DGX H100 and an RTX 3060. Prof-K is the fastest method in every large-N configuration on both GPUs, by up to $16\times$ over `torch.topk` and $3.1\times$ over RadiK, and its speed is unaffected by the input distribution, including inputs adversarial for radix selection. Its auxiliary memory is measured in megabytes where RadiK requires gigabytes, which extends exact selection to inputs on which radix selection cannot run, allowing to train larger models with larger batches on the same hardware.
- We demonstrate the utility of Prof-K in two machine learning workloads. In BatchTopK sparse autoencoders (Bussmann et al., 2024) trained on GPT-2 XL activations it reduces the end-to-end step time by 1.20 to $1.30\times$, close to the ceiling attainable by any selection kernel; in top-k gradient sparsification of GPT-2 XL it reduces the step time by $1.38\times$ with results identical to `torch.topk`. The same batch-global selection structure appears in Matryoshka and crosscoder variants of BatchTopK (Bussmann et al., 2025; Minder et al., 2025).

2 Related Work

Classical exact top-k methods are based on sorting, heaps, or partition-based selection such as quickselect, and efficient top-k query execution has long been studied in database and parallel

hardware settings (Akbarinia et al., 2011; Shanbhag et al., 2018). In large-scale GPU workloads the dominant cost is memory movement rather than comparisons, which makes exact top-k expensive even in optimized libraries such as PyTorch.

A substantial line of work studies efficient exact top-k selection on GPUs. Early work by Alabi et al. (2012) developed GPU-specific k-selection algorithms optimized for massively parallel architectures. Dr. Top-k (Gaihre et al., 2021) introduces a delegate-centric design that improves load balancing and reduces redundant work among GPU threads. Zhang et al. (2023) provide a comprehensive study of parallel GPU top-k algorithms and propose optimized implementations for different operating regimes. RTop-K (Xie et al., 2025) focuses on efficient row-wise top-k selection and demonstrates strong performance for moderate-size neural network workloads. Radix-based approaches such as RadiK (Li et al., 2024) further exploit floating-point representations to accelerate exact selection. In contrast, Prof-K reduces the size of the exact selection problem itself through probabilistic filtering, making it particularly effective for very large N and relatively small k.

Approximate and probabilistic selection methods reduce computational cost using sampling, sketching, or approximate quantile estimation, often at the cost of weaker guarantees or assumptions on the input distribution. Quantile summary methods such as the Greenwald–Khanna (Greenwald & Khanna, 2001) algorithm provide efficient streaming approximations with bounded error guarantees. Prof-K differs in that its guarantees depend only on the combinatorial properties of uniform sampling without replacement. The analysis therefore depends on ranks rather than values, yielding distribution-agnostic probabilistic guarantees.

Top-k selection is increasingly important in sparse machine learning workloads, including mixture-of-experts routing, sparse attention, activation pruning, and sparse autoencoders (Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022; Bussmann et al., 2024; 2025; Minder et al., 2025). Prof-K targets the subset of these workloads that perform batch-global selection over very large tensors with small retained fractions. Rather than replacing exact selection entirely, Prof-K uses sampling to construct a conservative threshold and then applies exact top-k refinement on a compact candidate buffer, making it complementary to optimized GPU kernels such as Dr. Top-k, RTop-K, and RadiK.

3 Problem Setup and Algorithm Overview

Given an input vector $x \in \mathbb{R}^N$ and an integer $k \geq 1$, our goal is to return the k largest entries of x together with their indices. For batched inputs with batch size B, the procedure is applied in parallel to each batch element.

The Prof-K Algorithm Prof-K proceeds in four stages, designed to minimize memory traffic while providing probabilistic correctness guarantees:

1. **Sampling.** Uniformly sample S indices without replacement from $[N] = \{1, \ldots, N\}$. Let $\mathcal{S} \subset [N]$ denote the sampled indices.
2. **Threshold Estimation.** Sort the sampled values $x|_{\mathcal{S}}$ in descending order. For a chosen rank $t \in [S]$, define the threshold $\tau = s_t$, where s_t is the t-th largest sampled value.
3. **Filtering.** Stream through the full input once, appending every element satisfying $x_i \geq \tau$ to a candidate buffer $\mathcal{B}$ of capacity M, which is usually small relative to the full input size N.
4. **Refinement.** Run an exact top-k routine on the buffer $\mathcal{B}$ to recover the final answer. If fewer than k candidates are collected, or if the buffer overflows before the scan completes, Prof-K redraws the sample and repeats the pass. Each attempt fails with probability at most ε independently of the previous ones, so the expected number of attempts is at most $1/(1-\varepsilon)$; an implementation may also fall back to an exact top-k over the full input after a fixed number of failed attempts, say six, an event of probability at most ε^6.

The central design goal is to choose the parameters (S, t, M) so that the number of retained elements $R = |\{i \in [N] : x_i \geq \tau\}|$ satisfies $k \leq R \leq M$ with probability at least $1 - \varepsilon$, where $0 < \varepsilon \ll 1$ is user specified. When this holds, the final exact top-k operates on a small buffer rather than all N entries. The idea behind the threshold is that the (t/S)-quantile of a uniform sample estimates the (t/S)-quantile of the input, so choosing t slightly above Sk/N yields a conservatively low threshold that all true top-k elements pass with high probability while most other elements are excluded.

4 THEORETICAL ANALYSIS

We now formalize the probabilistic guarantees underlying Prof-K. The analysis proceeds in four steps. First, we characterize the distribution of the rank that the sampled threshold occupies in the full input. Second, we prove a theorem that yields the exact optimal threshold rank $t^\star$ and buffer capacity $M^\star$ as functions of N, k, S and ε, without any approximation. Third, we give accurate and inexpensive approximations to these two quantities for settings in which the parameters must be recomputed frequently. Fourth, we derive an asymptotically optimal choice of the sample size S as a function of N and k that minimizes the computational overhead of the method. All proofs are given in Appendix A.

Notation and Conventions Throughout this section the input $x \in \mathbb{R}^N$ is fixed and its entries are assumed to be distinct, so that every entry has a well-defined rank, with rank 1 assigned to the largest entry. For $K \in \{0, 1, \dots, N\}$ we write $T_K \subset [N]$ for the set of indices of the K largest entries of x. The sampled index set $\mathcal{S} \subset [N]$ is drawn uniformly at random among all subsets of $[N]$ of size S; this is what sampling S indices uniformly without replacement means. As in Section 3, τ denotes the t-th largest sampled value and R its rank in the full input, so that $R = |\{i \in [N] : x_i \geq \tau\}|$ is the number of elements that pass the filter.

For integers $0 \leq K \leq N$ and $0 \leq S \leq N$, $X \sim \mathrm{Hypergeometric}(N, K, S)$ is the number of distinguished items among S items drawn uniformly without replacement from N items of which K are distinguished, that is,

$$\Pr(X = j) = \frac{\binom{K}{j}\binom{N-K}{S-j}}{\binom{N}{S}}, \qquad j \in \{0, 1, \dots, S\}, \tag{1}$$

with $\binom{a}{b} = 0$ whenever $b < 0$ or $b > a$. We write $F_{N,K,S}(j) = \Pr(X \leq j)$ for its distribution function and

$$F^{-1}_{N,K,S}(p) = \min\{\, j \in \{0, 1, \dots, S\} : F_{N,K,S}(j) \geq p \,\}, \qquad p \in (0, 1], \tag{2}$$

for its quantile function, which takes a probability and returns an integer; both are available in standard numerical libraries.

Sampled Counts in a Top-K Set The single quantity on which the analysis rests is the number of sampled indices that fall into a top-K set. For $K \in \{0, 1, \dots, N\}$ let $X_K = |\mathcal{S} \cap T_K|$, and define

$$G(K) = \Pr\big(\text{fewer than } t \text{ of the } S \text{ sampled indices lie in } T_K\big) = \Pr(X_K \leq t - 1). \tag{3}$$

The function G depends on N, S and t as well, and we make this dependence explicit by writing $G(N, K, S, t)$ whenever it matters.

Lemma 1. *For every $K \in \{0, 1, \dots, N\}$, $X_K \sim \mathrm{Hypergeometric}(N, K, S)$ and consequently*

$$G(K) = F_{N,K,S}(t - 1). \tag{4}$$

Moreover, $G(K)$ is nonincreasing in K and nondecreasing in t.

Failure Modes Exactly R elements pass the filter. If $R \geq k$, the k-th largest entry of x is at least τ, so every true top-k entry passes; if in addition $R \leq M$, every element that passes is stored, and the refinement step returns the exact top-k. Prof-K verifies the condition $k \leq R \leq M$ at run time, since the filter kernel counts the elements above the threshold exactly, and treats its violation as a failure. There are two ways in which this can happen:

1. **Insufficient recall**, $A = \{R < k\}$: the threshold is too high and fewer than k elements pass the filter.
2. **Buffer overflow**, $B = \{R > M\}$: more elements pass the filter than the buffer can hold, so some true top-k entries may fail to enter it.

Since $M \geq k$, the two events are disjoint and $\Pr(A \cup B) = \Pr(A) + \Pr(B)$. An overflow does not by itself make the output wrong, since the true top-k entries may still be among the M stored elements, so the probability of an incorrect output is bounded by $\Pr(A) + \Pr(B)$. We now compute both terms exactly.

Proposition 2 (Insufficient recall). *Let $1 \le k \le N$ and $1 \le t \le S$. Then for every input x,*

$$\Pr(R < k) = 1 - G(k-1) = 1 - F_{N,k-1,S}(t-1). \tag{5}$$

Proposition 3 (Buffer overflow). *Let $1 \le t \le S$ and $k \le M \le N$. Then for every input x,*

$$\Pr(R > M) = G(M) = F_{N,M,S}(t-1). \tag{6}$$

Neither probability depends on the values of x: only N, k, S, t and M enter Equations (5) and (6). This is the precise sense in which the guarantees of Prof-K are distribution-free.

Exact Optimal Parameters Given a failure budget for each mode, Propositions 2 and 3 reduce the choice of t and M to two inequalities in G, and the monotonicity established in Lemma 1 allows both to be solved exactly. The following theorem gives the smallest threshold rank and buffer capacity under which Prof-K retrieves the exact top-k with a user-specified probability on the first attempt.

Theorem 4 (Exact optimal parameters). *Let $1 \le k \le N$, $1 \le S \le N$ and $\varepsilon_A, \varepsilon_B \in (0,1)$ with $\varepsilon_A \ge \Pr(X_{k-1} = S)$, and set*

$$t^\star = F^{-1}_{N,k-1,S}(1-\varepsilon_A) + 1, \qquad M^\star = \min\big\{\, M \in \{k,\dots,N\} : F_{N,M,S}(t^\star - 1) \le \varepsilon_B \,\big\}. \tag{7}$$

Then $1 \le t^\star \le S$, the minimum defining $M^\star$ exists, and for every input x:

(i) $t^\star$ is the smallest threshold rank $t \in [S]$ for which the recall failure has probability $\Pr(R < k) \le \varepsilon_A$;

(ii) with $t = t^\star$, $M^\star$ is the smallest buffer capacity $M \ge k$ for which the overflow has probability $\Pr(R > M) \le \varepsilon_B$;

(iii) with $t = t^\star$ and $M = M^\star$, Prof-K passes its run-time check, and hence returns the exact top-k, with probability at least $1 - \varepsilon_A - \varepsilon_B$; the exact failure probability is

$$\Pr(A \cup B) = \Pr(R < k) + \Pr(R > M^\star). \tag{8}$$

The condition on ε_A only excludes a degenerate case: $X_{k-1} = S$ is the event that the entire sample falls within the top $k-1$ entries, and if it had probability above ε_A, no threshold rank $t \le S$ could meet the recall budget. Its probability $\binom{k-1}{S}/\binom{N}{S}$ is zero for $S \ge k$ and at most $((k-1)/N)^S$ otherwise, far below any budget of interest. With the default split $\varepsilon_A = \varepsilon_B = \varepsilon/2$, Theorem 4 states that Prof-K run with the parameters $(S, t^\star, M^\star)$ returns the exact top-k on the first attempt with probability at least $1-\varepsilon$, for every input. The two parameters are also jointly optimal in the following sense.

Corollary 5 (Joint optimality). *Under the assumptions of Theorem 4, every pair (t, M) with $t \in [S]$ and $M \in \{k,\dots,N\}$ satisfying $\Pr(R < k) \le \varepsilon_A$ and $\Pr(R > M) \le \varepsilon_B$ has $t \ge t^\star$ and $M \ge M^\star$.*

Computing the pair is inexpensive: $t^\star$ is one quantile evaluation, and since G is nonincreasing in M, $M^\star$ is found by binary search over $\{k,\dots,N\}$ with at most $\lceil \log_2 N \rceil$ evaluations of the distribution function.

Fast Approximate Parameter Computation Evaluating Equation (7) costs about a millisecond with a numerical library, comparable to the kernel itself. In most machine learning workloads the input shape and the sparsity level are fixed throughout training, so $(t^\star, M^\star)$ are computed once per configuration and cached; our implementation memoizes them. For configurations that change from call to call, Appendix B derives closed-form surrogates (a skewness-corrected normal or a Poisson quantile for $t^\star$, a Wilson–Hilferty quantile for $M^\star$) that need only elementary arithmetic, are conservative in every configuration tested, and lie within a few percent of $M^\star$; Algorithm 1 states the complete procedure.

Choice of the Sample Size Theorem 4 fixes $t^\star$ and $M^\star$ for a given S but leaves S free, and S controls a trade-off: a larger sample makes the threshold more accurate and the buffer smaller, at the price of more sampling work. We choose S by minimizing a cost model of the stages of Prof-K that charges the random gather of the S sampled elements and the selection of the top t among them against the exact selection over the $M^\star(S)$ buffer slots; the filtering pass streams the input once whatever S is. With the leading-order behavior $M^\star(S) \approx k + 2z\sqrt{kN/S} + \kappa N/S$, where

Algorithm 1 Fast approximate parameter computation (budget split $\varepsilon_A = \varepsilon_B = \varepsilon/2$)

Require: N, k, failure budget ε; sample size bounds $S_{\min}$, $S_{\max}$; calibrated cost ratio $r(\cdot)$
Ensure: sample size S, threshold rank t, buffer capacity M
1: $z \leftarrow \Phi^{-1}(1-\varepsilon/2)$ {rational approximation}
2: $\kappa \leftarrow z^2 + \frac{1}{2}$; $\quad r \leftarrow r(k)$ {calibrated cost ratio}
3: $S_1 \leftarrow (z/r)^{2/3}(kN)^{1/3}$; $\quad S_2 \leftarrow (\kappa N/r)^{1/2}$ {Proposition 7}
4: $S \leftarrow \mathrm{clip}\big(\lfloor \max(S_1, S_2) \rfloor,\ S_{\min},\ S_{\max}\big)$ {$S_{\min}$ if $r = \infty$ or $k/N > 2^{-10}$}
5: $\mu \leftarrow (k-1)S/N$
6: **if** $\mu \geq 15$ **then**
7: $\quad \sigma \leftarrow \sqrt{\mu\,(1 - \frac{k-1}{N})\,\frac{N-S}{N-1}}$; $\quad t \leftarrow \lceil \mu + z\sigma + (z^2-1)/6 \rceil + 1$ {Equation (10)}
8: **else**
9: $\quad p \leftarrow e^{-\mu}$; $\quad c \leftarrow p$; $\quad j \leftarrow 0$
10: $\quad$ **while** $c < 1 - \varepsilon/2$ **do**
11: $\quad\quad j \leftarrow j + 1$; $\quad p \leftarrow p\,\mu/j$; $\quad c \leftarrow c + p$ {Poisson quantile, Equation (11)}
12: $\quad$ **end while**
13: $\quad t \leftarrow j + 1$
14: **end if**
15: $h \leftarrow 1/(9t)$; $\quad M \leftarrow \left\lceil \frac{N+1}{S+1}\, t\,(1 - h + z\sqrt{h})^3 \right\rceil$ {Wilson–Hilferty, Equation (12)}
16: **return** $S,\ t,\ M$

$z = \Phi^{-1}(1-\varepsilon/2)$ for the standard normal distribution function Φ and $\kappa = z^2 + \frac{1}{2}$, the model has a closed-form minimizer up to a factor below 1.6 (Proposition 7 in Appendix B): the optimal sample size scales as $(kN)^{1/3}$ for large k and as $\sqrt{N}$ below, and its single hardware constant, the ratio r between the per-element costs of sampling and of selecting from the buffer, is calibrated once per GPU (Appendix E.1). The resulting rule, clipped to $[S_{\min}, S_{\max}]$ to reflect the fixed launch cost of the selection and to keep the sampling stage negligible against the scan, is given in lines 3 and 4 of Algorithm 1, which collects the complete parameter computation. Since the latency is flat over a wide range of S around the optimum (Appendix E.1), sweeping S and ε once for a fixed setup and caching the best values is an equally legitimate choice when the configuration does not change.

5 Experiments and Evaluation

We compare Prof-K against `torch.topk` and RadiK (Li et al., 2024), the state-of-the-art radix selection and the strongest exact baseline at large N, on a data-centre GPU (NVIDIA DGX H100, 80 GB) and a consumer GPU (NVIDIA RTX 3060 Laptop, 6 GB). Prof-K runs at $\varepsilon = 10^{-3}$ with parameters computed once per configuration and cached. The grid covers N from 2^{21} to 2^{30} and k from 2^5 to 2^{19} on the H100, extended to $N = 2^{31}$ against `torch.topk` alone, and N up to 2^{28} and k up to 2^{18} on the RTX 3060, whose memory limits the input size. Inputs are standard normal `float32` tensors; latencies are means over 30 input tensors and 30 timed calls each, measured with CUDA events after a warm-up, and auxiliary memory is the peak allocation above the baseline during one call. Other input distributions, including ones adversarial for radix selection, are covered in Appendix E.3, further details in Appendix D, and ablations on batched inputs in Appendix E.

Speed-up Across the (N, k) Space Figure 2 shows the speed-up over the full grid and Figure 3 the absolute latencies at $k = 2^{13}$. Three regimes are apparent. Below $N \approx 2^{23}$ every method takes a fraction of a millisecond, on the H100 between 0.2 and 0.5 ms at $k = 2^{13}$, and fixed costs dominate: Prof-K is on par with `torch.topk` and behind RadiK for small k on the H100, and ahead of both on the RTX 3060. From $N \approx 2^{24}$ the streaming pass dominates every method, and the advantage of a single bandwidth-bound pass becomes visible. On the H100 at $k = 2^{13}$ the latency of Prof-K grows from 0.25 ms at $N = 2^{24}$ to 1.7 ms at $N = 2^{30}$, against 0.45 to 26.6 ms for `torch.topk` and 0.42 to 5.4 ms for RadiK: the speed-up over `torch.topk` rises from 1.8× at 2^{24} to 5.2× at 2^{26}, 11× at 2^{28} and 15× at 2^{30}, and Prof-K is ahead of RadiK from $N = 2^{25}$ on in every configuration but two, reaching 3.1× at 2^{30}. On the RTX 3060 the picture is the same at a smaller scale: at $N = 2^{28}$ Prof-K takes 3.9 ms against 22.1 ms for `torch.topk` and 10.7 ms for RadiK, 5.7× and 2.7×. Finally, the dependence on k is weak: at fixed N the speed-up stays within a factor of about 1.3 of its small-k

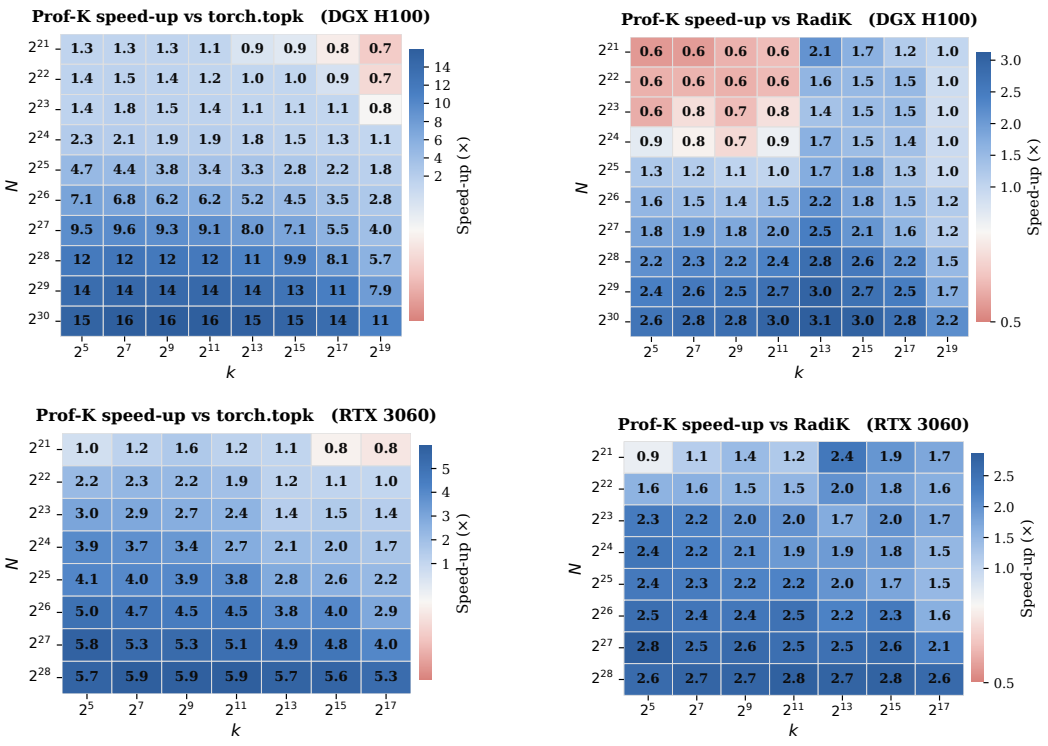


Figure 2: Speed-up of Prof-K over `torch.topk` (left) and RadiK (right) across the (N, k) grid at $\varepsilon = 10^{-3}$ on the DGX H100 (top) and the RTX 3060 (bottom). Blue marks configurations where Prof-K is faster, red where it is slower.

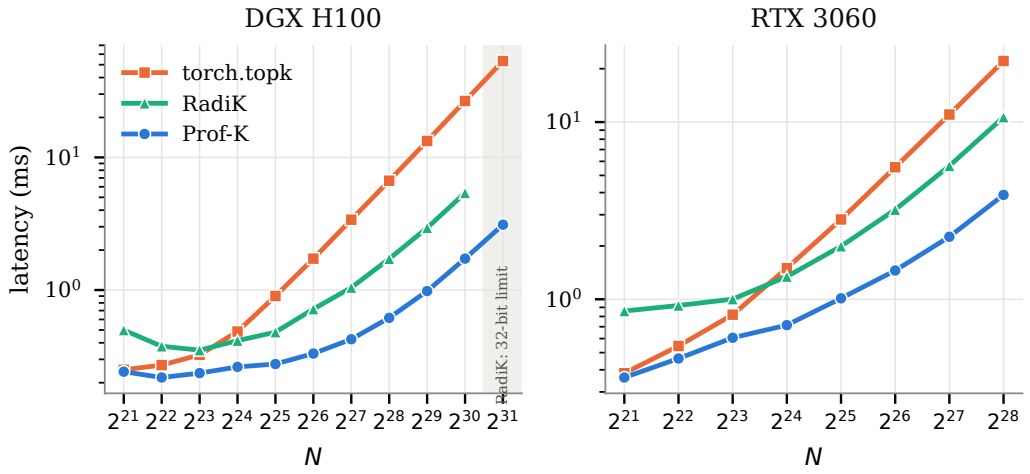


Figure 3: Latency as a function of the input size at $k = 2^{13}$ on both GPUs (log–log). Both baselines perform several passes over the input, Prof-K a single one, so its slope is that of the memory bandwidth. On the H100 the grid extends to $N = 2^{31}$ beyond the 32-bit limit of RadiK.

value up to $k \approx 2^{-10}N$ and declines only beyond that, where the buffer and the refinement grow with k.

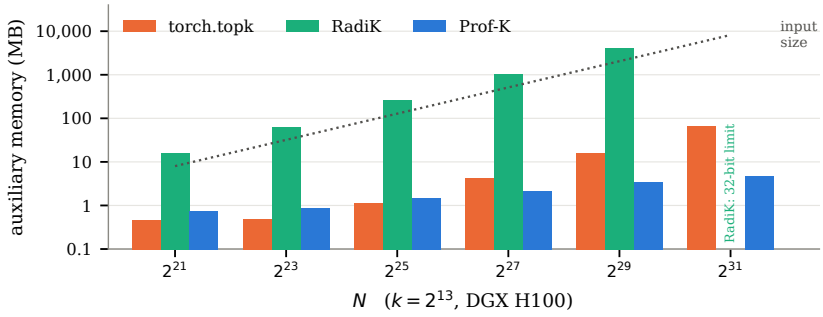


Figure 4: Auxiliary GPU memory on the DGX H100 at $k = 2^{13}$ (log scale); the dotted line is the size of the input. The footprints of `torch.topk` and RadiK grow with N, that of Prof-K with the buffer capacity $M^\star$.

Memory Footprint Figure 4 reports the auxiliary memory of the three methods on the H100 at $k = 2^{13}$. Both baselines are $O(N)$: RadiK allocates a workspace of about 8 bytes per input element, twice the size of the input itself, from 16 MB at $N = 2^{21}$ to 8 GB at $N = 2^{30}$, and `torch.topk` about 32 MB at 2^{30}. The workspace of RadiK is a structural property of its radix selection, which must keep per-element state across its passes. Prof-K allocates only the sample, the candidate buffer and the output, so its footprint is $O(M)$ rather than $O(N)$, and the measurements confirm it: from 0.8 MB at $N = 2^{21}$ to 4 MB at $N = 2^{30}$ at $k = 2^{13}$, growing with $M^\star$ and not with the input, and between 2 and 5 MB at $N = 2^{30}$ for every $k \le 2^{13}$, exceeding `torch.topk` only when k approaches 2^{20}. Prof-K therefore never runs out of memory where the input itself fits, and it keeps scaling where radix selection stops: at $N = 2^{31}$ it selects the top 2^{13} elements in 3.1 ms against 53.3 ms for `torch.topk`, a 17× speed-up, and it stays above 13× up to $k = 2^{19}$ and above 6× at $k = 2^{22}$ for inputs of $3 \cdot 10^9$ and $4.3 \cdot 10^9$ elements (Table 4 in Appendix D).

Application: BatchTopK Sparse Autoencoders To test whether kernel-level gains survive inside a training loop, we replace the batch-global top-k of BatchTopK sparse autoencoders (SAEs) (Bussmann et al., 2024) with Prof-K. At every step the post-encoder activations of a batch form a $B \times n$ matrix, n the dictionary size, flattened into $N = B \cdot n$ entries of which the largest $B \cdot k$ are retained: one selection per step over a pool of 10^8 to 10^9 elements with retained fraction k/n, on the critical path of training.

We train BatchTopK SAEs on residual-stream activations of GPT-2 XL (Radford et al., 2019) on a DGX H100 for 10^9 tokens, with dictionary sizes $n \in \{2^{16}, 2^{17}, 2^{18}\}$ and batch sizes B from 4,096 to 12,288, so that the pool $N = B \cdot n$ ranges from $2.7 \cdot 10^8$ to $2.1 \cdot 10^9$ elements (training details in Appendix D). The two runs of each configuration differ only in the selection call, `torch.topk` or Prof-K in exact mode, so the selected set and hence the training trajectory coincide, and the two runs are virtually identical in every tracked metric, as they should be. Table 1 reports the mean step time, the end-to-end speed-up and the GPU-hours of the full run. The speed-up grows with the pool size, from 1.20× at $2.7 \cdot 10^8$ to 1.30× at $2.1 \cdot 10^9$ elements, as the selection takes a growing share of the step, and it depends on N alone: the configurations $(2^{17}, 8192)$ and $(2^{18}, 4096)$ with the same N give the same gain, and the sparsity level does not enter, since top-k latency is nearly flat in k (Figure 2). In absolute terms Prof-K removes 0.6 to 2.1 GPU-hours from each 10^9-token run.

Application: Top-k Gradient Sparsification Top-k gradient sparsification typically communicates only the largest 0.1 to 1% of the gradient at every step and keeps the rest as an error-feedback residual (Lin et al., 2018). For a model with roughly 10^9 parameters this is one selection over 10^9 elements per step, and its cost has long been the main objection to the method: the selection has been measured at the same order as the backward pass or as the communication it is meant to save (Shi et al., 2019; Agarwal et al., 2022), and Yoon & Oh (2022) report uncompressed training achieving 4.67× higher throughput than exact Top-k sparsification on eight A100s, with the selection as the dominant cost. Several widely used remedies trade exactness for speed: thresholds estimated from a small sample (Lin et al., 2018) or from a Gaussian fit of the gradient distribution (Shi et al., 2019), or

Table 1: BatchTopK SAE training on GPT-2 XL activations (DGX H100, 10^9 tokens). $N = B \cdot n$ is the flattened pool size, ρ is the share of the baseline step spent in top-k, and the last two columns give the GPU-hours of the full run.

			step time (ms)				GPU-hours	
n	B	N	`torch.topk`	Prof-K	ρ	speed-up	`torch.topk`	Prof-K
2^{16}	4,096	$2.7 \cdot 10^8$	50.3	41.9	18.5%	1.20×	3.42	2.85
2^{17}	4,096	$5.4 \cdot 10^8$	71.4	58.4	22.4%	1.22×	4.84	3.96
2^{17}	6,144	$8.1 \cdot 10^8$	101.8	81.4	23.6%	1.25×	4.60	3.68
2^{17}	8,192	$1.1 \cdot 10^9$	132.2	104.4	24.2%	1.27×	4.48	3.54
2^{17}	12,288	$1.6 \cdot 10^9$	193.0	150.4	24.9%	1.28×	4.36	3.40
2^{18}	4,096	$1.1 \cdot 10^9$	142.8	115.0	22.4%	1.24×	9.69	7.80
2^{18}	6,144	$1.6 \cdot 10^9$	203.6	161.0	23.6%	1.26×	9.21	7.28
2^{18}	8,192	$2.1 \cdot 10^9$	264.4	203.5	25.1%	1.30×	8.97	6.91

Table 2: Single-H100 fine-tuning with error-feedback Top-k gradient sparsification at 0.1% density. Both selectors return the same Top-k set at every step.

		step time (ms)		
Model	Parameters	`torch.topk`	Prof-K	speed-up
GPT-2 Medium	355M	85	73	1.16×
GPT-2 XL	1.56B	172	125	1.38×

thresholds or index sets periodically recomputed and reused across steps (Li & Hoefler, 2022; Zheng & Zhang, 2025). The exact primitive is still expensive: interpolating the large-N H100 measurements reported in Table 4 to the $1.56 \cdot 10^9$ parameters of GPT-2 XL at 0.1% density gives estimated selection latencies of about 40 ms for `torch.topk` and 4.5 ms for Prof-K. Prof-K nevertheless returns the exact top-k set at every step. Failures of the sampled filter are detected and handled, so the returned set remains exact. Exact per-step selection is therefore practical in this regime and avoids both the estimation risk of the threshold-based schemes, whose selected count can fall far short of k and stall convergence when the gradient distribution departs from the assumed one (Abdelmoniem et al., 2021), and the resampling interval of the index-reuse schemes, whose extremes degrade the loss (Zheng & Zhang, 2025). Fine-tuning GPT-2 XL with error-feedback top-k sparsification at 0.1% density on a single H100 confirms this in the full training loop; Table 2 shows a $1.38\times$ step-time speed-up for GPT-2 XL and $1.16\times$ for GPT-2 Medium, with identical selected sets at every step.

6 Conclusion

We introduced Prof-K, a probabilistic one-pass filtering method for exact top-k selection on GPUs. It estimates a threshold from a small sample, streams once through the input into a compact candidate buffer, and selects the top-k from this buffer, while a run-time check guarantees that every output is exact. A distribution-free analysis gives the failure probability exactly and yields the smallest parameters for a prescribed budget. On a DGX H100, Prof-K is up to $16\times$ faster than `torch.topk` and $3.1\times$ faster than RadiK, requires only a few megabytes of auxiliary memory, and scales to inputs that radix selection cannot process. At GPT-2 XL scale, it reduces the training time of BatchTopK sparse autoencoders and top-k gradient sparsification, making it a practical building block for sparse machine learning workloads.

Limitations Prof-K offers no advantage on small inputs, where the fixed cost of its four stages is comparable to that of a full exact selection and RadiK remains slightly faster unless many rows are batched. Moreover, the sample-size rule depends on a cost ratio calibrated once per GPU, and the method has not been evaluated on multi-GPU inputs, which we leave to future work.

AI USE STATEMENT

In this work, we used large language models for two purposes, both among the tasks for which disclosure is recommended: to refine the text of the paper for clarity, and to search for some of the related work. We have reviewed all AI-assisted work: every suggested reference was checked against its source, and every edited passage was read and approved by the authors.

ETHICS STATEMENT

This work improves the efficiency of a general computational primitive and involves no human subjects, personal data or data collection; all experiments use synthetic inputs, public models and public datasets. The authors do not foresee negative ethical or societal impacts specific to this contribution beyond those of the applications in which top-k selection is used, and declare no conflict of interest.

REPRODUCIBILITY STATEMENT

We release the code of Prof-K, comprising the Triton filter kernel, the exact and approximate parameter computation and core function implementations, as anonymized supplementary material, with a public release to follow. The Triton kernel itself is reproduced in Appendix C. The assumptions of every theoretical result are stated in Section 4, the proofs are given in Appendix A, and the parameter approximations are derived in Appendix B and summarized in Algorithm 1. The hardware, the benchmark grid and the measurement protocol of the evaluation are described in Section 5, with further implementation details of the evaluation and of every ablation in Appendices D and E.

REFERENCES

Ahmed M. Abdelmoniem, Ahmed Elzanaty, Mohamed-Slim Alouini, and Marco Canini. An efficient statistical-based gradient compression technique for distributed training systems. In *Proceedings of Machine Learning and Systems*, volume 3, pp. 297–322, 2021.

Peter J. Acklam. An algorithm for computing the inverse normal cumulative distribution function. Archived copy: `https://web.archive.org/web/20151030215612/http://home.online.no/~pjacklam/notes/invnorm/`.

Saurabh Agarwal, Hongyi Wang, Shivaram Venkataraman, and Dimitris Papailiopoulos. On the utility of gradient compression in distributed training systems. In *Proceedings of Machine Learning and Systems*, volume 4, pp. 652–672, 2022.

Reza Akbarinia, Esther Pacitti, and Patrick Valduriez. Best position algorithms for efficient top-k query processing. *Information Systems*, 36(6):973–989, 2011.

Tolu Alabi, Jeffrey D Blanchard, Bradley Gordon, and Russel Steinbach. Fast k-selection algorithms for graphics processing units. *Journal of Experimental Algorithmics (JEA)*, 17:4.1–4.29, 2012.

Bart Bussmann, Patrick Leask, and Neel Nanda. Batchtopk sparse autoencoders. *arXiv preprint arXiv:2412.06410*, 2024.

Bart Bussmann, Noa Nabeshima, Adam Karvonen, and Neel Nanda. Learning multi-level features with Matryoshka sparse autoencoders. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 6077–6101, 2025.

Felix Chern, Blake Hechtman, Andy Davis, Ruiqi Guo, David Majnemer, and Sanjiv Kumar. Tpu-knn: K nearest neighbor search at peak flop/s. In *Advances in Neural Information Processing Systems*, volume 35, pp. 15489–15501, 2022.

Edmund A Cornish and Ronald A Fisher. Moments and cumulants in the specification of distributions. *Revue de l'Institut International de Statistique*, 5(4):307–320, 1938.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Anil Gaihre, Da Zheng, Scott Weitze, Lingda Li, Shuaiwen Leon Song, Caiwen Ding, Xiaoye S Li, and Hang Liu. Dr. top-k: delegate-centric top-k on gpus. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–14, 2021.

Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. *ACM SIGMOD Record*, 30(2):58–66, 2001.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=qrwe7XHTmYb.

Shigang Li and Torsten Hoefler. Near-optimal sparse allreduce for distributed deep learning. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 135–149, 2022.

Yifei Li, Bole Zhou, Jiejing Zhang, Xuechao Wei, Yinghan Li, and Yingda Chen. Radik: scalable and optimized gpu-parallel radix top-k selection. In *Proceedings of the 38th ACM International Conference on Supercomputing*, pp. 537–548, 2024.

Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J. Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SkhQHMW0W.

Julian Minder, Clément Dumas, Caden Juang, Bilal Chughtai, and Neel Nanda. Overcoming sparsity artifacts in crosscoders to interpret chat-tuning. In *Advances in Neural Information Processing Systems*, volume 38, pp. 106423–106474, 2025.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 2019.

Anil Shanbhag, Holger Pirk, and Samuel Madden. Efficient top-k query processing on massively parallel hardware. In *Proceedings of the 2018 International Conference on Management of Data*, pp. 1557–1570, 2018.

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*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Shaohuai Shi, Xiaowen Chu, Ka Chun Cheung, and Simon See. Understanding top-k sparsification in distributed deep learning. *arXiv preprint arXiv:1911.08772*, 2019.

Edwin B. Wilson and Margaret M. Hilferty. The distribution of chi-square. *Proceedings of the National Academy of Sciences*, 17(12):684–688, 1931.

Xi Xie, Yuebo Luo, Hongwu Peng, and Caiwen Ding. Rtop-k: Ultra-fast row-wise top-k selection for neural network acceleration on gpus. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/ca1b93fc0f3560ba84eb0bc8de6d8f91-Paper-Conference.pdf.

Daegun Yoon and Sangyoon Oh. Empirical analysis on top-k gradient sparsification for distributed deep learning in a supercomputing environment. *arXiv preprint arXiv:2209.08497*, 2022.

Jingrong Zhang, Akira Naruse, Xipeng Li, and Yong Wang. Parallel top-k algorithms on gpu: A comprehensive study and new methods. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–13, 2023.

Mingkai Zheng and Zhao Zhang. Radius: Range-based gradient sparsity for large foundation model pre-training. In *Proceedings of Machine Learning and Systems*, volume 7, 2025.

A PROOFS

Proof of Lemma 1. All $\binom{N}{S}$ subsets of size S are equally likely, and exactly $\binom{K}{j}\binom{N-K}{S-j}$ of them contain precisely j indices of T_K, which gives Equation (1) for X_K; Equation (4) follows from Equation (3). Monotonicity in t is immediate from the definition of a cumulative distribution function. For $K < K'$ we have $T_K \subset T_{K'}$, hence $X_K \le X_{K'}$ pointwise and $\Pr(X_{K'} \le t-1) \le \Pr(X_K \le t-1)$. □

Proof of Proposition 2. The event $\{R < k\}$ is the event that τ is one of the $k-1$ largest entries of x. Since τ is the t-th largest sampled value, this happens if and only if the t largest sampled values all belong to T_{k-1}, that is, if and only if $X_{k-1} \ge t$. Hence $\Pr(R < k) = \Pr(X_{k-1} \ge t) = 1 - \Pr(X_{k-1} \le t-1) = 1 - G(k-1)$, and Lemma 1 gives the second equality. □

Proof of Proposition 3. The event $\{R > M\}$ is the event that τ is not among the M largest entries of x, which happens if and only if fewer than t sampled indices belong to T_M, that is, $X_M \le t-1$. Hence $\Pr(R > M) = G(M)$, and Lemma 1 gives the second equality. □

Proof of Theorem 4. Write $F = F_{N,k-1,S}$. By Proposition 2, $\Pr(R < k) \le \varepsilon_A$ holds if and only if $F(t-1) \ge 1 - \varepsilon_A$. Since F is nondecreasing, the set of such t is $\{t : t - 1 \ge F^{-1}(1-\varepsilon_A)\}$, whose smallest element is $t^\star$; this proves (i). The quantile $F^{-1}(1-\varepsilon_A)$ is at most S, and it equals S only if $F(S-1) < 1 - \varepsilon_A$, that is, $\Pr(X_{k-1} = S) > \varepsilon_A$, which the assumption excludes; hence $t^\star \le S$. For (ii), by Proposition 3 the condition $\Pr(R > M) \le \varepsilon_B$ reads $G(M) \le \varepsilon_B$ with $t = t^\star$. By Lemma 1, G is nonincreasing in M, so the set of $M \in \{k, \dots, N\}$ satisfying it is an interval ending at N; it is nonempty because $X_N = S \ge t^\star$ gives $G(N) = 0$. Its smallest element is $M^\star$ by definition. Finally, A and B are disjoint because $M^\star \ge k$, which gives Equation (8), and the bound $\varepsilon_A + \varepsilon_B$ follows from (i) and (ii). □

Proof of Corollary 5. $t \ge t^\star$ is (i) of Theorem 4. Since G is nondecreasing in t by Lemma 1, $G(N, M, S, t^\star) \le G(N, M, S, t) \le \varepsilon_B$, so M satisfies the defining condition of $M^\star$ and $M \ge M^\star$ by minimality. □

Proof of Proposition 7. $\tilde{C}$ is a sum of a linear function and two strictly convex decreasing functions of S, so it is strictly convex, and $\tilde{C}(S) \to \infty$ as $S \to 0$ or $S \to \infty$, so the minimizer exists and is unique. It is the root of $\tilde{C}'(S) = 0$, that is, of $r = f_1(S) + f_2(S)$ with $f_1(S) = z\sqrt{kN}\, S^{-3/2} = r(S_1/S)^{3/2}$ and $f_2(S) = \kappa N S^{-2} = r(S_2/S)^2$, both decreasing. At $S = \max(S_1, S_2)$ one of f_1, f_2 equals r, so $f_1 + f_2 \ge r$ and $S^\circ \ge \max(S_1, S_2)$. At $S = 2^{2/3}\max(S_1, S_2)$ we have $f_1 \le r/2$ and $f_2 \le 2^{-4/3} r \le r/2$, so $f_1 + f_2 \le r$ and $S^\circ \le 2^{2/3}\max(S_1, S_2)$. □

Proof of Lemma 6. Write the sampled indices in increasing order of rank as $\rho_1 < \rho_2 < \cdots < \rho_S$, so that $\rho_t = R$. The S sampled ranks split the $N - S$ unsampled ranks into $S + 1$ gaps of sizes $D_1, \dots, D_{S+1} \ge 0$, where D_i is the number of unsampled ranks strictly between ρ_{i-1} and ρ_i (with $\rho_0 = 0$ and $\rho_{S+1} = N + 1$), and $\rho_t = t + \sum_{i \le t} D_i$. Since $\mathcal{S}$ is a uniformly random subset of size S, the vector $(D_1, \dots, D_{S+1})$ is uniformly distributed over the compositions of $N - S$ into $S + 1$ nonnegative parts, a distribution that is invariant under permutations of the parts; hence the D_i are exchangeable and $\mathbb{E}[D_i] = (N - S)/(S + 1)$ for every i. Therefore

$$\mathbb{E}[R] = t + t\,\frac{N - S}{S + 1} = \frac{N + 1}{S + 1}\,t. \qquad \square$$

B FAST PARAMETER COMPUTATION

This appendix derives the closed-form surrogates for $t^\star$ and $M^\star$ that Algorithm 1 in Section 4 uses, derives the sample-size rule of Section 4, and compares the surrogates with the exact parameters of Theorem 4. Throughout, $\varepsilon_A = \varepsilon_B = \varepsilon/2$ and $z = \Phi^{-1}(1 - \varepsilon/2)$, where Φ is the standard normal distribution function; Φ^{-1} is itself evaluated by the rational approximation of Acklam, whose relative error is below $1.15 \cdot 10^{-9}$ over the whole domain, so that the fast path has no library dependence.

Each surrogate is chosen according to the regime in which it is accurate, and each is designed to err on the side of a larger t or a larger M, since an underestimate of either would violate its budget.

Threshold rank. By Theorem 4, $t^\star - 1$ is the $(1-\varepsilon_A)$-quantile of $X_{k-1} \sim \mathrm{Hypergeometric}(N, k-1, S)$, whose mean and variance are

$$\mu = \frac{(k-1)S}{N}, \qquad \sigma^2 = \mu\Big(1 - \frac{k-1}{N}\Big)\frac{N-S}{N-1}. \tag{9}$$

The quantity μ, the expected number of sampled indices among the top-$(k-1)$, determines the shape of the distribution and hence the appropriate approximation. When $\mu \geq 15$, X_{k-1} is close to normal, but it remains right-skewed, and the symmetric normal quantile $\mu + z\sigma$ systematically underestimates its upper tail. We therefore use the second-order Cornish–Fisher expansion (Cornish & Fisher, 1938), which corrects the normal quantile of a distribution with skewness γ by the term $(z^2-1)\gamma\sigma/6$. For the hypergeometric distribution $\gamma\sigma = (N-2(k-1))(N-2S)/(N(N-2)) \leq 1$, with equality approached in the sparse regime $k \ll N$, $S \ll N$; replacing $\gamma\sigma$ by its upper bound 1 keeps the correction conservative. This gives

$$t = \Big\lceil \mu + z\sigma + \frac{z^2-1}{6} \Big\rceil + 1 \qquad (\mu \geq 15). \tag{10}$$

When $\mu < 15$, the normal approximation is unreliable, but in this regime $k - 1 \ll N$ and X_{k-1} is a count of rare events, for which $\mathrm{Poisson}(\mu)$ is the natural limit: $\mathrm{Hypergeometric}(N, k-1, S)$ converges to $\mathrm{Poisson}(\mu)$ as $N \to \infty$ with $(k-1)S/N = \mu$ fixed. Since $\mathrm{Poisson}(\mu)$ has variance $\mu \geq \sigma^2$, its upper quantiles dominate those of X_{k-1}, and replacing one by the other is again conservative. We therefore take $t = 1 + j^\star$ with

$$j^\star = \min\Big\{ j \geq 0 : \sum_{i=0}^{j} e^{-\mu}\frac{\mu^i}{i!} \geq 1 - \varepsilon_A \Big\} \qquad (\mu < 15), \tag{11}$$

evaluated by accumulating the terms through the recurrence $p_0 = e^{-\mu}$, $p_i = p_{i-1}\mu/i$, which stops after $O(\mu + z\sqrt{\mu})$ iterations, that is, a few dozen at most. For $k = 1$ this gives $\mu = 0$ and $t = 1$, which is exact: the sample maximum never exceeds the global maximum, so $\Pr(R < 1) = 0$.

Buffer capacity. By Proposition 3, $M^\star$ is the smallest M with $\Pr(R > M) \leq \varepsilon_B$, that is, the $(1-\varepsilon_B)$-quantile of the rank R itself. Its distribution is that of the t-th smallest of S positions drawn uniformly without replacement from $[N]$, and its mean is known exactly.

Lemma 6. *For* $1 \leq t \leq S \leq N$, $\mathbb{E}[R] = \dfrac{N+1}{S+1}\,t$.

As $N \to \infty$ with S fixed, the sampled positions become independent uniform points, and $R/(N+1)$ converges in distribution to the t-th order statistic of S independent $\mathrm{Uniform}(0,1)$ variables; as $S \to \infty$ with t fixed, this order statistic rescaled by $S+1$ converges to $\mathrm{Gamma}(t,1)$. We therefore approximate R by $\frac{N+1}{S+1}\Gamma_t$ with $\Gamma_t \sim \mathrm{Gamma}(t,1)$, a scaling that matches $\mathbb{E}[R]$ exactly by Lemma 6. This choice retains the feature that matters most for an upper quantile: for small t the distribution of R is strongly right-skewed, and a normal approximation with the same mean and variance would underestimate $M^\star$ severely, whereas $\mathrm{Gamma}(t,1)$ has the same skew. To evaluate the quantile of Γ_t in closed form we use the Wilson–Hilferty transform (Wilson & Hilferty, 1931), by which $(\Gamma_t/t)^{1/3}$ is approximately normal with mean $1-h$ and variance h, where $h = 1/(9t)$. This yields

$$M = \Big\lceil \frac{N+1}{S+1}\, t\,\big(1 - h + z\sqrt{h}\big)^3 \Big\rceil, \qquad h = \frac{1}{9t}, \tag{12}$$

for every t. For large t the cube expands to $t + z\sqrt{t} + O(1)$, the normal quantile of $\mathrm{Gamma}(t,1)$, so Equation (12) reduces to the normal approximation exactly where the latter is valid. At the other extreme $t = 1$, where Γ_1 is exponential with $(1-\varepsilon_B)$-quantile $\ln(1/\varepsilon_B)$, Equation (12) overestimates the quantile by about 3% for $\varepsilon_B = 5 \times 10^{-4}$, which is the conservative direction. The complete procedure, combining Equations (10) to (12) with the choice of S derived next, is summarized in Algorithm 1.

Choice of the Sample Size Theorem 4 fixes $t^\star$ and $M^\star$ for a given S but leaves S free, and S controls a trade-off: a larger sample makes the threshold more accurate and the buffer smaller, at the price of more sampling work. We choose S by minimizing a cost model of the stages of Prof-K on a GPU. The filtering pass streams the N elements once at memory bandwidth and does not depend on S or M. The other stages process a number of elements proportional to S or M: drawing the sample is a random gather of S elements at a per-element cost g, well above the streaming cost since each random access fetches a full memory sector; the threshold is a top-t selection over the S sampled values at a per-element cost b; and the refinement is a top-k selection over the M buffer slots at a per-element cost $d(k)$ that grows with k, since selecting many elements from the buffer is closer to a sort than to a scan. Fixed launch overheads do not affect the minimizer. The overhead beyond the mandatory scan is therefore

$$C(S) = (g + b)\, S + d(k)\, M^\star(S), \tag{13}$$

where $M^\star(S)$ denotes the exact buffer capacity of Theorem 4 at sample size S. To minimize Equation (13) in closed form we use the leading-order behavior of $M^\star(S)$, which follows from the surrogates above:

$$M^\star(S) \;\approx\; k + 2z\sqrt{\frac{kN}{S}} + \kappa\,\frac{N}{S}, \qquad \ln(1/\varepsilon_B) \le \kappa \le z^2 + \tfrac{1}{2}, \tag{14}$$

where the lower end of the range for κ is attained at $t = 1$ and the upper end for large t; we use $\kappa = z^2 + \frac{1}{2}$. The square-root term is the familiar sampling error of a quantile estimate; the N/S term is the contribution of the skew of R at small t, and it dominates in the sparse regime $\mu \lesssim 1$, where the buffer is governed by N/S rather than by k. Substituting Equation (14) into Equation (13) and writing $r = (g + b)/d(k)$ for the cost of sampling one element relative to selecting one from the buffer, the model has a unique minimizer, which we characterize as follows.

Proposition 7 (Sample size)**.** *Let $r, z, \kappa > 0$ and let $\tilde{C}(S) = rS + 2z\sqrt{kN/S} + \kappa N/S$ for $S > 0$. Then $\tilde{C}$ is strictly convex with a unique minimizer S°, and with*

$$S_1 = \left(\frac{z}{r}\right)^{2/3}(kN)^{1/3}, \qquad S_2 = \left(\frac{\kappa N}{r}\right)^{1/2}, \tag{15}$$

it satisfies $\max(S_1, S_2) \le S^\circ \le 2^{2/3}\max(S_1, S_2)$.

The two candidates correspond to the two terms of Equation (14): for $k \gtrsim z\sqrt{rN}$ the optimal sample size scales as $(kN)^{1/3}$, and below that as $\sqrt{N}$, independently of k, because the buffer is then governed by N/S. We use $S = \mathrm{clip}(\max(S_1, S_2), S_{\min}, S_{\max})$, which by Proposition 7 is within a factor $2^{2/3} \approx 1.59$ of the model optimum, and the sensitivity study in Appendix E.1 shows that the latency is flat over a considerably wider range of S around it. The lower bound $S_{\min}$ reflects the fixed launch cost of the selection routine, below which a smaller sample saves nothing, and $S_{\max}$ keeps the sampling stage negligible against the scan. The only hardware-dependent quantity is the ratio r, which we calibrate once per GPU from a sweep over S (Appendix E.1). The calibration confirms the two regimes the model predicts: for k up to a few thousand the buffer selection costs less per element than the sampling, r is large, and the rule returns the floor $S_{\min}$; for larger k the buffer selection dominates, $1/r$ grows linearly in $\log_2 k$, and the sample grows with k and N. When the retained fraction exceeds $k/N \approx 2^{-10}$ the threshold selection over the sample, whose rank t grows with Sk/N, becomes the dominant cost as well, and the floor is again the best choice.

Accuracy Table 3 compares the parameters of Algorithm 1 with the exact optimal ones of Theorem 4 at $\varepsilon = 10^{-3}$ over 21 configurations spanning N from 2^{20} to 2^{31} and k from 2^5 to 2^{20}, with the sample size of Algorithm 1 in both cases. For each pair of parameters the table also reports the exact failure probability $\Pr(R < k) + \Pr(R > M)$, evaluated with the hypergeometric distribution function of Propositions 2 and 3. The surrogate threshold rank equals $t^\star$ in 19 configurations and exceeds it by one in the other two; the surrogate buffer capacity exceeds $M^\star$ by 0.1 to 3%, with the largest excess at $t = 1$, where the Wilson–Hilferty quantile is least accurate, and the excess vanishes as t grows. The attained failure probability is below the budget in every configuration for both parameter sets, and below that of the exact parameters for the surrogates, which confirms that the approximations err on the conservative side.

Table 3: Fast approximate parameters (Algorithm 1) against the exact optimal parameters (Theorem 4) at $\varepsilon = 10^{-3}$, with the exact failure probability $\Pr(R < k) + \Pr(R > M)$ attained by each pair.

N	k	S	$t^\star$	t	$M^\star$	M	$M/M^\star$	Pr(fail), exact	Pr(fail), approx.
2^{20}	2^5	8,192	4	4	1,777	1,801	1.014	$5.99 \cdot 10^{-4}$	$5.28 \cdot 10^{-4}$
2^{20}	2^8	8,192	9	9	2,835	2,855	1.007	$7.09 \cdot 10^{-4}$	$6.60 \cdot 10^{-4}$
2^{20}	2^{11}	8,192	32	32	6,883	6,909	1.004	$7.53 \cdot 10^{-4}$	$7.09 \cdot 10^{-4}$
2^{20}	2^{14}	8,192	167	168	27,141	27,385	1.009	$9.70 \cdot 10^{-4}$	$7.64 \cdot 10^{-4}$
2^{24}	2^5	16,384	2	2	10,232	10,422	1.019	$9.35 \cdot 10^{-4}$	$8.57 \cdot 10^{-4}$
2^{24}	2^8	16,384	4	4	14,259	14,402	1.010	$6.28 \cdot 10^{-4}$	$5.75 \cdot 10^{-4}$
2^{24}	2^{11}	20,650	10	10	19,281	19,362	1.004	$7.91 \cdot 10^{-4}$	$7.60 \cdot 10^{-4}$
2^{24}	2^{14}	36,403	58	58	39,777	39,822	1.001	$8.27 \cdot 10^{-4}$	$8.09 \cdot 10^{-4}$
2^{24}	2^{17}	16,384	168	168	218,748	219,092	1.002	$8.87 \cdot 10^{-4}$	$8.49 \cdot 10^{-4}$
2^{28}	2^5	16,384	2	2	163,769	166,737	1.018	$5.02 \cdot 10^{-4}$	$4.26 \cdot 10^{-4}$
2^{28}	2^8	37,480	3	3	86,297	87,385	1.013	$5.07 \cdot 10^{-4}$	$4.47 \cdot 10^{-4}$
2^{28}	2^{11}	82,603	5	5	51,044	51,435	1.008	$9.89 \cdot 10^{-4}$	$9.45 \cdot 10^{-4}$
2^{28}	2^{14}	110,643	18	18	85,613	85,777	1.002	$7.37 \cdot 10^{-4}$	$7.19 \cdot 10^{-4}$
2^{28}	2^{17}	234,231	152	153	224,429	225,793	1.006	$9.45 \cdot 10^{-4}$	$8.20 \cdot 10^{-4}$
2^{28}	2^{20}	16,384	93	93	2,095,476	2,098,101	1.001	$8.83 \cdot 10^{-4}$	$8.60 \cdot 10^{-4}$
2^{31}	2^5	16,384	1	1	996,031	1,026,231	1.030	$7.36 \cdot 10^{-4}$	$6.34 \cdot 10^{-4}$
2^{31}	2^8	106,011	2	2	202,532	206,164	1.018	$5.78 \cdot 10^{-4}$	$5.03 \cdot 10^{-4}$
2^{31}	2^{11}	233,638	4	4	128,067	129,279	1.009	$5.86 \cdot 10^{-4}$	$5.35 \cdot 10^{-4}$
2^{31}	2^{14}	262,144	9	9	181,989	182,719	1.004	$7.37 \cdot 10^{-4}$	$7.08 \cdot 10^{-4}$
2^{31}	2^{17}	262,144	32	32	441,824	442,225	1.001	$7.76 \cdot 10^{-4}$	$7.65 \cdot 10^{-4}$
2^{31}	2^{20}	262,144	168	168	1,752,499	1,752,830	1.000	$9.09 \cdot 10^{-4}$	$9.04 \cdot 10^{-4}$

C Implementation Details

The Filter Kernel The only custom kernel in Prof-K is the filtering pass, shown in Listing 1; sampling, the threshold selection over the sample and the final selection over the buffer are standard library calls (`torch.topk` on the S sampled values and on the M buffer slots). The kernel is written in Triton and launched on a two-dimensional grid, one axis over blocks of `BLOCK_SIZE` consecutive elements of a row and one over the rows of the batch, so that batched inputs are handled by the same code with a per-row threshold, buffer and counter. Each program loads its block once, compares it against the threshold of its row, and counts the elements that pass. If any do, it reserves a contiguous range of buffer slots for them with a single atomic addition on the per-row counter, computes the position of each retained element within that range by an in-block prefix sum, and writes values and indices to the reserved slots. The pass therefore streams the input exactly once, issues one atomic per block rather than one per retained element, and writes only the retained elements, which is what keeps it at the memory bandwidth of the device.

The counter is incremented by the full number of retained elements in the block, whereas the stores are masked to the buffer capacity. After the pass the counter of each row therefore holds the exact rank R of its threshold, whether or not the buffer overflowed, and the run-time check $k \le R \le M$ of Section 3 is a comparison of B integers, without any extra pass over the input. The block size and the number of warps are selected by Triton's autotuner from the configurations listed, keyed on the row length, and the counters are reset to zero between autotuning trials so that the trials do not accumulate into them.

D Experimental Details and Additional Results

Latency of Every Configuration Figure 5 reports the absolute latency of the three methods on every (N, k) configuration of the grid on both GPUs, grouped by N with k increasing within each group.

Speed-up as a Function of N Figure 6 shows every (N, k) configuration of the grid of Section 5 as one point, with the spread over k made visible. Table 4 lists the configurations beyond $N = 2^{30}$

```python
import triton
import triton.language as tl


@triton.autotune(
    configs=[
        triton.Config({'BLOCK_SIZE': 1024}, num_warps=8, num_stages=3),
        triton.Config({'BLOCK_SIZE': 2048}, num_warps=8, num_stages=4),
        triton.Config({'BLOCK_SIZE': 4096}, num_warps=8, num_stages=4),
    ],
    key=['n_elements'],
    reset_to_zero=['count_ptr'],
)
@triton.jit
def _filter_topk_batched_kernel(
    x_ptr, out_vals_ptr, out_idx_ptr, count_ptr, thresholds_ptr,
    stride_xb: tl.int64, stride_ob: tl.int64, n_elements: tl.int64,
    buffer_cap,
    BLOCK_SIZE: tl.constexpr,
):
    pid_chunk = tl.program_id(axis=0)   # block of the input row
    pid_batch = tl.program_id(axis=1)   # row of the batch

    # 64-bit offsets: inputs may exceed 2^31 elements
    block_start = pid_chunk.to(tl.int64) * BLOCK_SIZE
    offsets = block_start + tl.arange(0, BLOCK_SIZE)
    read_mask = offsets < n_elements

    x_offset = pid_batch.to(tl.int64) * stride_xb + offsets
    x = tl.load(x_ptr + x_offset, mask=read_mask, other=-float('inf'))
    thresh = tl.load(thresholds_ptr + pid_batch)

    pass_mask = (x >= thresh) & read_mask
    pass_int = pass_mask.to(tl.int32)
    n_pass = tl.sum(pass_int)

    if n_pass > 0:
        # one atomic per block reserves n_pass slots and counts every
        # retained element, also beyond the buffer capacity
        global_offset = tl.atomic_add(count_ptr + pid_batch, n_pass)
        local_offset = tl.cumsum(pass_int) - 1
        write_idx = global_offset + local_offset
        write_mask = pass_mask & (write_idx < buffer_cap)

        out_offset = pid_batch * stride_ob + write_idx
        tl.store(out_vals_ptr + out_offset, x, mask=write_mask)
        tl.store(out_idx_ptr + out_offset, offsets.to(tl.int64),
                 mask=write_mask)
```

Listing 1: The Triton filter kernel of Prof-K.

on the H100, where RadiK cannot run because its 32-bit indexing stops at 2^{31} elements, so Prof-K is compared with `torch.topk` alone.

Cost of the Run-Time Check Exact mode is guaranteed to return the true top-k; the price is the host synchronisation needed to inspect the count R after the filtering pass, and the rare redraw. On the H100 this costs between 5% of the latency at $N = 2^{30}$ and 40% at $N = 2^{21}$, where the synchronisation exposes kernel-launch latency that is otherwise hidden. Omitting the check, which is appropriate when an occasional miss is acceptable, recovers this overhead, and we study the resulting accuracy–speed trade-off in Appendix E.

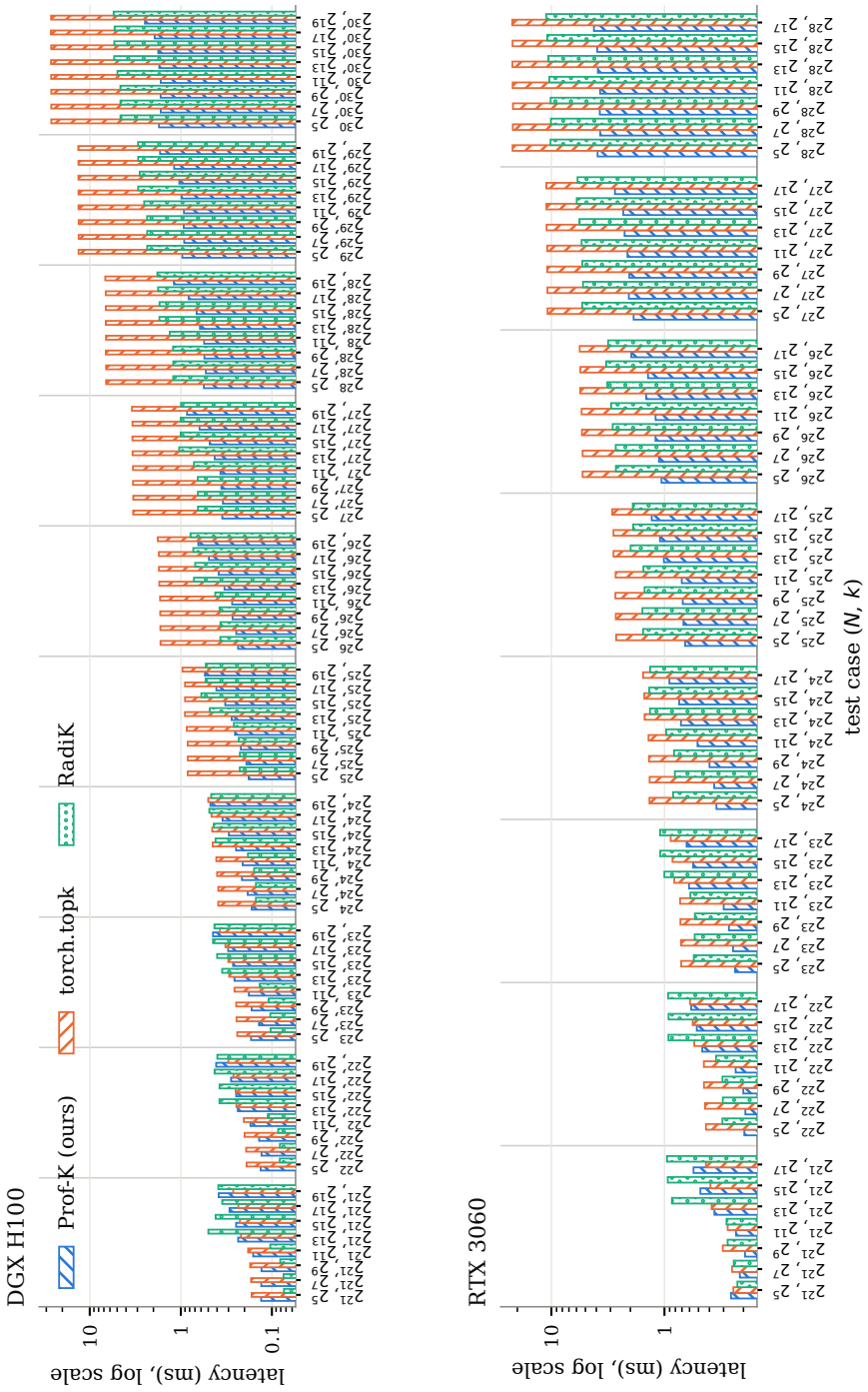


Figure 5: Latency (ms, log scale) of `torch.topk`, RadiK and Prof-K on every (N, k) configuration of the grid, on the DGX H100 (top, $k \in \{2^5, 2^7, \ldots, 2^{19}\}$) and the RTX 3060 (bottom, $k \in \{2^5, 2^7, \ldots, 2^{17}\}$), at $\varepsilon = 10^{-3}$. Groups are ordered by N and, within a group, by k.

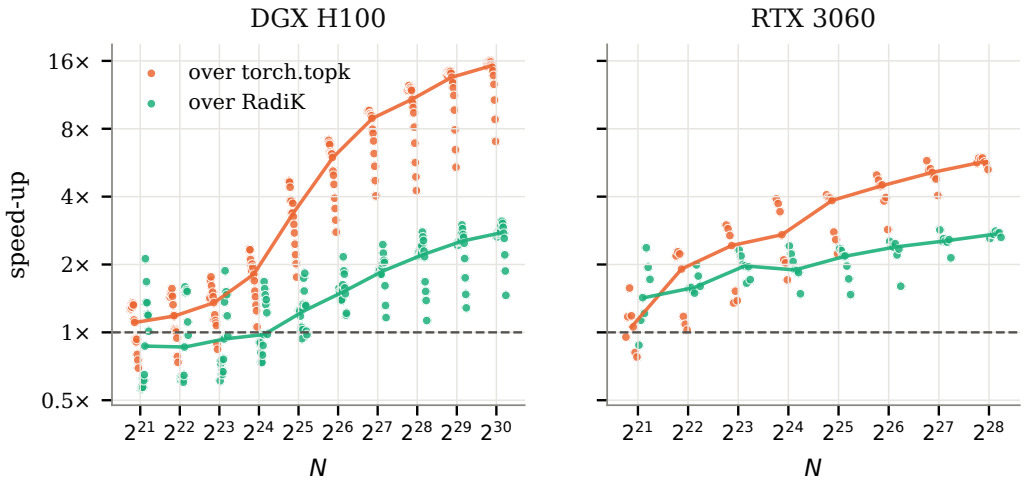


Figure 6: Speed-up of Prof-K as a function of N; each point is one (N, k) configuration, horizontally spread by k, and the lines join the medians over k. The dashed line marks parity.

Table 4: Prof-K against `torch.topk` on the DGX H100 for inputs beyond the reach of RadiK. Latency in ms, auxiliary memory in MB.

N	k	Prof-K (ms)	`torch.topk` (ms)	speed-up	Prof-K (MB)	`torch.topk` (MB)
2^{31}	2^{10}	3.14	53.8	17.1×	3.1	65.0
2^{31}	2^{14}	3.19	53.5	16.8×	5.8	65.2
2^{31}	2^{18}	3.56	53.4	15.0×	24.1	68.0
2^{31}	2^{22}	7.25	54.0	7.5×	238	177
$3 \cdot 10^9$	2^{10}	4.67	74.6	16.0×	5.4	91.4
$3 \cdot 10^9$	2^{18}	5.61	74.6	13.3×	32.4	94.4
$4.3 \cdot 10^9$	2^{10}	6.61	106.7	16.1×	5.7	130.0
$4.3 \cdot 10^9$	2^{18}	7.51	106.8	14.2×	35.1	133.0
$4.3 \cdot 10^9$	2^{22}	15.4	106.5	6.9×	285	178

E Ablations

E.1 Sample Size

The sample size S is the one parameter of Prof-K that the exact analysis of Section 4 leaves free, and Proposition 7 chooses it from a cost model whose single hardware constant, the cost ratio r, has to be calibrated. We therefore sweep S on the RTX 3060 to calibrate r and to measure how far the resulting rule lands from the latency-optimal sample size. The sweep covers $S \in \{2^{12}, 2^{13}, \ldots, 2^{18}\}$, capped at $N/32$, for every $N \in \{2^{21}, \ldots, 2^{28}\}$ and $k \in \{2^5, 2^8, 2^{11}, 2^{14}, 2^{17}\}$; for each S the threshold rank and buffer capacity are the exact optimal ones of Theorem 4 at $\varepsilon = 10^{-3}$, and the latency is that of exact mode, averaged over 8 input tensors and 20 timed calls each after a warm-up.

Calibration Fitting the cost model $C = c + (g + b)S + d(k)M^\star(S)$ to the 40×5 measurements with $S \geq 2^{14}$, with one intercept c per configuration, one sampling cost $g + b$, and one buffer cost $d(k)$ per value of k, explains the latencies to within 4.6% on average ($R^2 = 0.998$). The sampling cost is 0.36 ns per element. The buffer cost is not resolvable from zero for $k \leq 2^{11}$, and equals 1.6 ns per element at $k = 2^{14}$ and 2.4 ns at $k = 2^{17}$: selecting many elements from the buffer costs several times more per element than sampling, which is why the sample should grow with k. The ratio is well described by $1/r(k) = 0.59 \log_2 k - 4.3$ for $k > 2^7$ and $r = \infty$ below, and this is the function $r(\cdot)$ used in Algorithm 1.

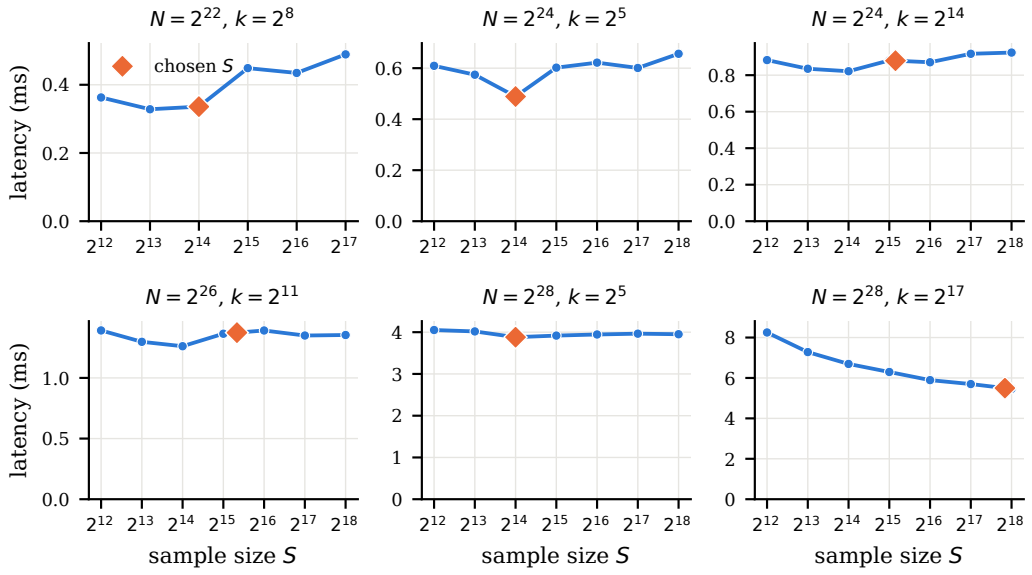


Figure 7: Exact-mode latency of Prof-K as a function of the sample size S on the RTX 3060 for six (N, k) configurations, with the sample size chosen by Algorithm 1 marked. The curves are flat over most of the range; at large k and large N a larger sample pays for itself through a smaller buffer, and the rule follows it.

Table 5: Sample-size ablation on the RTX 3060 (exact mode, $\varepsilon = 10^{-3}$). For each N, mean over the five values of k of the ratio between the slowest and the fastest sample size in the sweep, and of the latency at the sample size chosen by Algorithm 1 relative to the empirical optimum.

N	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}	2^{26}	2^{27}	2^{28}
slowest / fastest S	1.33	1.40	1.34	1.20	1.11	1.13	1.20	1.29
Algorithm 1 / optimum	1.01	1.00	1.01	1.02	1.04	1.04	1.05	1.01

Sensitivity and Accuracy of the Rule Figure 7 shows six representative configurations and Table 5 summarizes all 40. First, the latency is insensitive to S over a wide range: across the $64\times$ span of the sweep, the ratio between the slowest and the fastest sample size has a median of 1.24 and never exceeds 1.51, so a factor of two in either direction around the optimum typically costs a few percent. Second, the empirical optimum is $S = 2^{14}$ in 27 of the 40 configurations, which is the fixed-cost floor $S_{\min}$; the exceptions are the large-k configurations, where the optimum moves to 2^{18} as N grows. Third, the sample size returned by Algorithm 1 lands on average 2.2% above the optimal latency of the sweep, with a worst case of 15% for moderate k at large N, where the rule leaves the floor slightly earlier than the measurements warrant; run-to-run variation between sessions on this GPU is of the same order, so we do not resolve the rule further.

E.2 Batched Inputs

The main evaluation concerns a single large vector. Here we ask whether Prof-K also pays off on batched inputs, where each of B rows of length N is selected independently in one call, and what relaxing the failure budget buys. We measure `torch.topk`, RadiK and Prof-K in approximate mode on the RTX 3060 for $N \in \{2^{16}, 2^{18}, 2^{20}, 2^{22}\}$, $B \in \{1, 16, 128, 1024\}$ with $B \cdot N \le 2^{28}$, and $k \in \{64, 1024\}$, on standard normal inputs of shape (B, N), averaging over 5 input tensors and 20 timed calls each after a warm-up. The budget ε is per call and is split as ε/B across the rows, so that by the union bound the probability that any row misses is at most ε; the recall, the fraction of the true top-k indices returned, is measured against `torch.topk`.

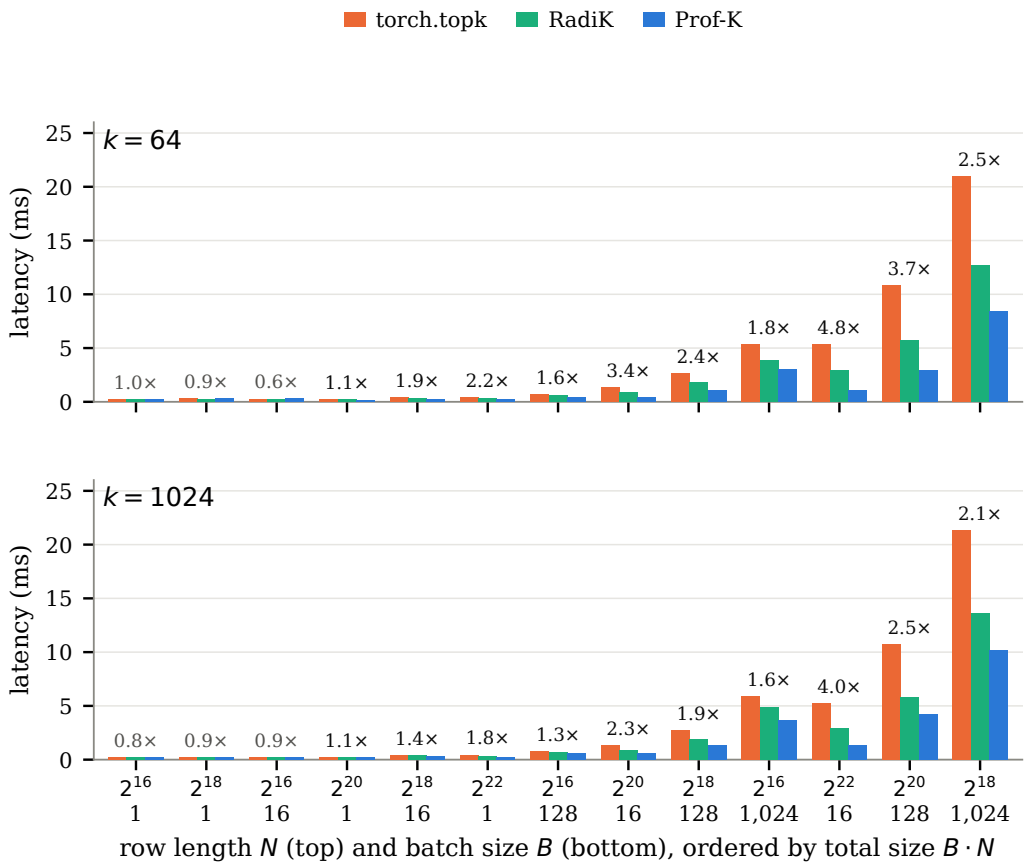


Figure 8: Latency of `torch.topk`, RadiK and Prof-K (approximate mode, $\varepsilon = 10^{-3}$) on batched inputs on the RTX 3060, for $k = 64$ (top) and $k = 1024$ (bottom). Each group is one (N, B) configuration, ordered by the total number of elements $B \cdot N$; the number above a group is the speed-up of Prof-K over `torch.topk`.

Batching Extends the Gains to Small N Figure 8 shows the latencies at $\varepsilon = 10^{-3}$. For a single row of $N \leq 2^{20}$ no method takes more than 0.5 ms and the three are within 10 to 40% of each other, since the fixed costs of the four stages of Prof-K are comparable to an exact selection over a million elements. Batching amortizes these fixed costs, and the total size $B \cdot N$ is what governs the outcome. From $B \cdot N = 2^{23}$ on, Prof-K is ahead of both baselines in every configuration, and its advantage grows with the total size: $3.4\times$ over `torch.topk` and $2.1\times$ over RadiK at $N = 2^{20}$, $B = 16$; $4.8\times$ and $2.6\times$ at $N = 2^{22}$, $B = 16$; $3.7\times$ and $2.0\times$ at $N = 2^{20}$, $B = 128$; and $2.5\times$ and $1.5\times$ at $N = 2^{18}$, $B = 1024$, all for $k = 64$, with $k = 1024$ between 10 and 30% lower. At a fixed total size, longer rows are more favourable than more rows, since the per-row costs of the threshold selection and of the buffer are then spread over more elements: $B \cdot N = 2^{26}$ gives $4.8\times$ for $N = 2^{22}$, $B = 16$ but $1.8\times$ for $N = 2^{16}$, $B = 1024$. The crossover at $B \cdot N \approx 2^{22}$ coincides with the one for a single vector in Figure 2 once the horizontal axis there is read as the total size. The recall is 1.000 in every configuration.

Relaxing the Budget Table 6 reports the same measurements at four budgets. Loosening ε from 10^{-3} to 0.3 shortens the latency by only about 5% at the median, because S, $t^\star$ and $M^\star$ depend on ε only through $z = \Phi^{-1}(1 - \varepsilon/2)$, whereas the recall of single rows drops to between 0.87 and 0.97. On batches the split budget keeps the per-row budget small even at $\varepsilon = 0.3$, and the recall stays above 0.996. Prof-K therefore has no useful accuracy knob below the exact regime, and there is no

reason to run it with a loose budget: the recommended operating point is $\varepsilon = 10^{-3}$, at which the approximate mode returns the exact top-k in all but one call in a thousand.

Table 6: Approximate mode of Prof-K on the RTX 3060 as a function of the failure budget ε, over the 26 configurations of Figure 8: latency relative to $\varepsilon = 10^{-3}$ (median), and minimum and median recall for single rows ($B = 1$) and for batches ($B \geq 16$).

ε	10^{-3}	10^{-2}	0.1	0.3
latency relative to $\varepsilon = 10^{-3}$	1.00	0.95	0.93	0.95
recall, $B = 1$ (min / median)	1.000 / 1.000	1.000 / 1.000	0.966 / 0.995	0.871 / 0.945
recall, $B \geq 16$ (min / median)	1.000 / 1.000	1.000 / 1.000	0.998 / 1.000	0.997 / 1.000

E.3 Adversarial Input Distributions

The guarantee of Section 4 depends only on the ranks induced by uniform sampling, never on the values, so the latency of Prof-K should be the same for every input distribution. Radix selection is not distribution-free: it discards candidates one digit of the binary representation at a time, and inputs whose values share their leading bits defeat the reduction. Li et al. (2024) name values concentrated in a narrow range, such as uniform on $[128.6, 128.7]$, as an adversarial case for RadiK and counter it with an adaptive scaling step that subtracts a randomly drawn element before the selection. We measure all methods on the RTX 3060 at $N = 2^{27}$ and $k \in \{2^8, 2^{13}\}$ on seven input distributions: this adversarial case, uniform on $[1, 2)$, whose values share their sign and exponent bits, and five benign ones, including the heavy-tailed log-normal and Cauchy. RadiK is run without its scaling step, as in the rest of the paper, and with it; Prof-K runs in exact mode at $\varepsilon = 10^{-3}$, and every returned value is verified against `torch.topk`.

Table 7: Latency (ms) on the RTX 3060 at $N = 2^{27}$ as a function of the input distribution: `torch.topk`, RadiK without and with (†) its adaptive scaling step, and Prof-K in exact mode at $\varepsilon = 10^{-3}$. Mean over 5 input tensors and 20 timed calls each.

	$k = 2^8$				$k = 2^{13}$			
input distribution	`torch`	RadiK	RadiK†	Prof-K	`torch`	RadiK	RadiK†	Prof-K
standard normal	11.2	5.3	5.3	2.3	11.5	5.8	5.8	2.4
uniform $[0, 1)$	11.6	6.1	5.6	2.2	11.3	6.6	6.1	2.4
uniform $[1, 2)$	11.2	6.9	5.7	2.2	11.4	7.3	6.2	2.3
uniform $[128.6, 128.7]$	10.8	15.8	5.6	2.2	11.2	16.4	6.0	2.4
log-normal	11.3	5.3	5.3	2.3	11.6	5.8	5.8	2.4
exponential	11.7	5.4	5.3	2.2	11.4	6.0	5.8	2.5
Cauchy	11.4	5.4	5.4	2.3	11.7	5.8	5.8	2.4

Table 7 confirms the picture. Prof-K takes 2.2 to 2.5 ms on every distribution, with at most 0.02 redraws per call and exact values on every call. RadiK without scaling slows down as soon as values share their leading bits, by 15 to 30% on uniform inputs and by 3× on the narrow-range input, where it falls behind `torch.topk`; the speed-up of Prof-K over it grows accordingly, from 2.3× on normal inputs to 7× on the adversarial one. Its scaling step removes the worst case, after which Prof-K is still 2.3 to 2.6× faster on every distribution. Heavy tails cost no method anything, since the top-k elements then occupy their own leading digits. Prof-K is thus impervious to the input distribution in practice as well as in theory, and it needs no input-dependent preprocessing to be so.

E.4 Failure Budget

The failure budget ε is the one user-facing parameter of Prof-K. In exact mode it trades the cost of a single attempt, which grows slowly as ε shrinks through $z = \Phi^{-1}(1 - \varepsilon/2)$ in the sample size and the buffer capacity, against the frequency of redraws, which grows linearly in ε. We measure this trade-off on the RTX 3060 in exact mode with the exact parameters of Theorem 4, for ε from 0.3 down to 10^{-4} at $N \in \{2^{24}, 2^{28}\}$ and $k \in \{2^8, 2^{14}\}$, recording for every call the count R of the first attempt, the number of redraws and the latency of the whole call including the check. Between 2,000 and 10,000 calls are made per configuration, 140,000 in total.

The Failure Probabilities Are Exact The count R of the first attempt tells which failure mode occurred, so the experiment measures $\Pr(R < k)$ and $\Pr(R > M)$ directly. Table 8 compares the observed numbers of recall failures and buffer overflows, pooled over the four (N, k) configurations, with the numbers predicted by Propositions 2 and 3 at the same parameters. The two agree at every budget within the sampling noise of the counts, from 1,194 overflows observed against 1,200 predicted at $\varepsilon = 0.3$ to 15 against 20 at 10^{-3}, and the observed total never exceeds ε times the number of calls. The recall failures are rarer than ε_A allows at small k, where the integer threshold rank cannot exhaust its budget: at $N = 2^{28}$, $k = 2^8$ and $\varepsilon = 0.3$ the rank is $t^\star = 1$ and $\Pr(R < k)$ is 0.015 rather than 0.15.

Table 8: Predicted and observed numbers of recall failures ($R < k$) and buffer overflows ($R > M$) on the first attempt, pooled over the four (N, k) configurations of the failure-budget ablation (RTX 3060, exact mode). Predictions are the exact probabilities of Propositions 2 and 3 multiplied by the number of calls.

ε	calls	predicted $R < k$	observed $R < k$	predicted $R > M$	observed $R > M$
0.3	8,000	545.8	518	1,199.9	1,194
0.1	8,000	248.0	245	399.9	415
0.03	8,000	41.5	34	120.0	116
0.01	12,000	31.4	38	60.0	51
0.003	24,000	16.5	25	36.0	30
10^{-3}	40,000	7.0	7	20.0	15
10^{-4}	40,000	1.0	0	2.0	2

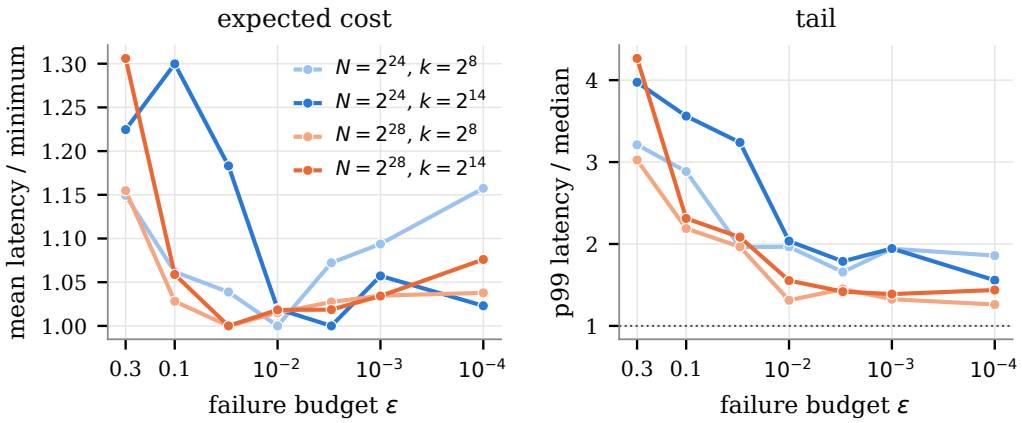


Figure 9: Exact-mode latency of Prof-K on the RTX 3060 as a function of the failure budget ε, per call including the run-time check and any redraw. Left: mean latency relative to its minimum over ε. Right: the 99th percentile relative to the median.

The Expected Cost Is Flat, but the Tail Events Are Not Figure 9 shows the latency. The mean is within 10% of its minimum for every ε between 10^{-3} and 10^{-2} in all four configurations, with the minimum between 0.003 and 0.03: the larger buffer of a tight budget and the redraws of a loose one cost about the same there. Outside that range both ends lose, by up to 16% at $\varepsilon = 10^{-4}$, where the buffer at $N = 2^{24}$, $k = 2^8$ is 34% larger than at 10^{-3}, and by 15 to 30% at $\varepsilon = 0.3$, where a call redraws 0.2 to 0.4 times on average. The tail separates the two ends: for $\varepsilon \le 10^{-2}$ the 99th percentile stays within the run-to-run jitter of this GPU, 1.3 to 2× the median, whereas at $\varepsilon = 0.1$ one call in ten runs the pass twice and the 99th percentile rises to 2.2 to 3.6× the median, and to 3 to 4.3× at $\varepsilon = 0.3$. A budget of 10^{-3} therefore costs at most 10% in expectation against the fastest budget and keeps the latency of every call predictable, which is why we recommend it as a reasonable default.