Qrita: High-performance Top-k and Top-p Algorithm for GPUs using Pivot-based Truncation and Selection

Jongseok Park
University of California, Berkeley
js_park@berkeley.edu

Sunga Kim
University of California, Berkeley
sunga.kim@berkeley.edu

Alvin Cheung
University of California, Berkeley
akcheung@berkeley.edu

Ion Stoica
University of California, Berkeley
istoica@berkeley.edu

Abstract

Despite their importance in model sampling, efficient implementation of the Top-k and Top-p algorithms poses a significant challenge as model vocabulary grows with each new model release. Existing approaches often rely on sorting, which incurs significant computation and memory overhead on GPUs, or on stochastic approaches that alter the algorithm's output. In this work, we propose Qrita, an efficient Top-k and Top-p algorithm based on pivot-based truncation and selection. Qrita leverages pivot-based search for both Top-k and Top-p with two key techniques: 1. Gaussian-based σ-truncation, which **limits the complexity of the algorithm to** $\mathcal{O}(n)$, and 2. Quaternary pivot search with duplication handling, which halves the number of pivot search iterations and **guarantees deterministic output**. We implement Qrita using Triton and evaluate its performance against the Top-k and Top-p kernels of high-performance LLM execution engines such as SGLang and FlashInfer, improving end-to-end serving throughput up to 1.4$\times$ with half the memory usage, while providing the **same output as stable sort-based Top-k and Top-p algorithms**. Qrita is now the default Triton Top-k and Top-p sampler of vLLM, and its implementation is available at `https://github.com/vllm-project/vllm/blob/main/vllm/v1/sample/ops/topk_topp_triton.py`.

1 Introduction

Top-k and Top-p truncation are two key mechanisms used in sampling-based generation of Large Language Models (LLMs) to balance output quality and diversity. Top-k [5] selects the k highest-logit tokens, pruning low-probability options to prevent incoherent output. In contrast, Top-p [10] selects the smallest subset whose cumulative probability exceeds a threshold p, offering a dynamic and diversity-oriented cutoff. Together, Top-k and Top-p provide a robust solution that stabilizes LLM generation while preserving contextual variation.

Despite their conceptual simplicity, implementing efficient Top-k and Top-p algorithms on GPUs remains a significant challenge. Modern LLM execution engines, such as vLLM [12], rely on full-vocabulary sort-and-filter methods to identify the target Top-k and Top-p subset. Such sorting-based algorithms provide exact results but are computationally inefficient on GPUs. Divide-and-conquer solutions such as the Bitonic sort [19] provide ways to utilize the massively parallel architecture of GPUs, but still incur significant memory overhead as repeated comparisons between array elements require strided memory accesses and substantial buffer space for intermediate results.

Recent research explores alternative directions to mitigate these inefficiencies. FlashInfer [27] eliminates sorting overhead via iterative *rejection sampling*, designed to be stochastically equivalent

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

to Top-k and Top-p sampling. However, because it returns a single token with only probabilistic guarantees, it is incompatible with tasks that require the full deterministic candidate set, such as speculative decoding verification [13] and reinforcement learning [15, 7]. Other solutions, such as Top-$n\sigma$ [21] and Min-p [16], propose statistical cutoffs based purely on the logit distribution. Statistical cutoffs remove sorting entirely, allowing for highly efficient single-pass $\mathcal{O}(n)$ algorithms. However, the generation quality of heuristic methods is highly debated [20], and standard Top-k and Top-p sampling remain the dominant method in production environments [12, 28].

Thus, a critical gap remains for a Top-k and Top-p algorithm that delivers both memory efficiency and strict deterministic behavior on the ever-increasing LLM vocabulary. To this end, we investigate RTop-k [24], a pivot-based truncation strategy originally designed for node selection in Graph Neural Networks. RTop-k offers a memory-efficient alternative to sorting using an in-place binary search for a Top-k **truncation pivot**, a value where exactly k elements in the set are larger than or equal to it. However, RTop-k is not directly applicable to LLM sampling for two reasons: (1) its search algorithm is optimized for small vectors ($<$1,000 elements) and becomes inefficient over the large LLM vocabularies often exceeding 100,000 elements; and (2) it lacks explicit handling for duplicate logits, which can trigger excessive search iterations and non-deterministic behavior.

To address these limitations, we propose Qrita, a high-performance Top-k and Top-p algorithm for GPUs. Qrita extends the concept of pivot-based truncation of RTop-k to be compatible with the joint Top-k and Top-p pipeline in LLM decoding, employing two key techniques to achieve computational efficiency and determinism: (1) **Gaussian σ-truncation**, limits the which *complexity of the algorithm to* $\mathcal{O}(n)$ by leveraging the stochastic distribution of the logit values and the high sparsity of Top-k and Top-p operations in LLM sampling, and (2) **Quaternary pivot search with duplication handling**, which halves the search iterations while guaranteeing determinism with *exact same output as stable sort-based Top-k and Top-p algorithms* using multi-pivot bookkeeping of duplicate logit values, allowing fast and reproducible outputs.

In summary, Qrita provides fast and deterministic Top-k/Top-p via truncation, pivot search, and duplication handling, with **three key advantages** over existing algorithms. (1) Qrita avoids full-vocabulary sorting, reducing memory overhead and providing high performance on GPUs. (2) Qrita returns the full Top-k/Top-p candidate set instead of a single sample token, retaining compatibility with downstream tasks that require the full set. (3) Qrita retains exact Top-k/Top-p semantics and reproducibility, unlike statistical cutoff methods that alter the sampling output.

We implement Qrita using Triton [23], a popular GPU programming language. We provide ablation studies of Qrita and perform extensive evaluations against the state-of-the-art solutions provided by vLLM [12], SGLang [28], and FlashInfer [27], demonstrating throughput improvements up to 1.4$\times$ while providing the same output as the stable sorting-based Top-k and Top-p algorithms.

2 Related Work

Top-k and Top-p Sampling in LLMs: Top-k [5] and Top-p (nucleus) sampling [10] are the standard mechanisms for controlling the randomness of LLM generation. Top-k restricts the candidate pool to the k most probable tokens, effectively filtering out the long tail of low-probability tokens. Top-p refines this approach by dynamically truncating the tail based on the cumulative probability mass p, allowing the candidate set size to expand or contract based on the model's confidence. These operators are typically implemented by sorting the entire vocabulary (often $> 100,000$ tokens), truncating the top candidates, and/or calculating the cumulative sum of probabilities [12]. While true to the definition of Top-k and Top-p, this sorting-based approach is inefficient on the massively parallel architecture of GPUs and creates a significant memory overhead in the form of strided memory access and intermediate output buffers.

High-performance libraries such as FlashInfer [27] provide sorting-free Top-k and Top-p sampling by leveraging Dual Pivot Rejection Sampling, an algorithm that iteratively refines the inverse transform probability range with rejection sampling until a token is sampled [25]. While this design returns a stochastically equivalent result to the original Top-k and Top-p-based sampling, it fundamentally alters the operator's interface: it returns a *single sampled index* rather than a set of candidates. This limitation renders it inadequate for advanced decoding tasks that require inspecting the candidate set, such as speculative decoding verification [13] or semantic caching systems [3], where a known "best" set is required. Furthermore, as the parallel prefix-sum operations required for inverse transform

Algorithm 1 High-level Qrita overview for Top-k

1: **Input:** Batched logits $\mathbf{Z}$, Top-k target $\mathbf{K}$, Vocab size $\mathcal{V}$, Batch size B
2: **Output:** Top-k output $\mathbf{O}$
3: **for** b in $0 \dots B-1$ **do** // Parallelized as Triton Programs
4: $\quad\mathbf{Z}_t \leftarrow \text{GAUSSIANSIGMATRUNCATION}(\mathbf{Z}[b], \mathcal{V})$ // Truncate Gaussian noise region
5: $\quad$**if** $|\mathbf{Z}_t| > \mathbf{K}[b]$ **then** // Truncation Hit: Truncated set large enough
6: $\quad\quad\mathbf{Z}_s \leftarrow \mathbf{Z}_t$ // Search truncated set
7: $\quad$**else** // Truncation Miss: Not enough elements
8: $\quad\quad\mathbf{Z}_s \leftarrow \mathbf{Z}[b]$ // Search original full logit set
9: $\quad$**end if**
10: $\quad\tau, z_{dup}, N_{dup} \leftarrow \text{QUATERNARYSEARCH}(\mathbf{Z}_s, \mathbf{K}[b])$ // Search pivot and find duplicates
11: $\quad\mathbf{O}[b] \leftarrow \text{DUPLICATIONHANDLING}(\mathbf{Z}, \tau, z_{dup}, N_{dup})$ // Resolve duplicates for exact Top-k
12: **end for**

sampling are non-associative in floating-point arithmetic, non-deterministic behavior and numerical instability may arise when executed across different hardware configurations [25].

Statistical Threshold Methods: An alternative direction, explored by methods such as η-sampling [8], Top-$n\sigma$ [21], and Min-p [16], is to eliminate sorting by relying on the statistical properties of the logit distribution. These approaches observe that LLM logits typically follow a quasi-Gaussian distribution, where a large number of "noisy" tokens cluster around the mean and a small number of "informative" tokens form a high-probability outlier group. By calculating simple statistics like the mean (μ) and standard deviation (σ) of the distribution, these methods define a hard cutoff threshold (e.g., $\tau = \mu + n\sigma$) to separate the signal from the noise in a single pass. While computationally efficient, these statistical methods produce a completely different sampling output compared to standard Top-k and Top-p, leading to debated effects on generation coherence [20]. Our work leverages this statistical intuition not as a replacement, but as a *pre-filtering* step (Gaussian σ-truncation) to Top-k and Top-p to accelerate the algorithm without altering the final output.

Pivot-Based Selection Algorithm: Quickselect [9] and more recent algorithms such as RadixSelect [2] and RTop-k [24] replace sorting with a search and select of a k-th pivot value. By iteratively selecting a pivot value τ and counting the number of elements $x_i \geq \tau$, the algorithm can narrow down the range containing the k-th largest element without moving data. This approach is highly effective for small vector sizes where the data fits in fast on-chip memory. However, for LLMs with massive vocabularies, the repeated passes across the logit set on each iteration saturate the memory bandwidth, making it slower than optimized solutions such as that of FlashInfer. Furthermore, standard pivot-based methods are non-deterministic in the presence of duplicate logits often found in long-tailed distributions with low precision, as the search cannot detect and distinguish indices with duplicate values. Qrita addresses these limitations by using a quaternary search to reduce memory passes and bookkeeping of duplicate logits to ensure determinism.

3 Algorithm Design

We present the design of Qrita using Top-k as the main example, and highlight how the Top-p algorithm differs from Top-k. Algorithm 1 provides a high-level overview of Qrita. The `for` loop across the batch dimension (line 3) is parallelized as Triton Programs[1] [22], and operations inside each batch are parallelized as Triton blocks. We use a persistent kernel design and launch $\min(B, N_{sm})$ Triton programs, where N_{sm} is the number of multiprocessors on the GPU. In the following, we skip the batch dimension subscripts when explaining the operations within each batch for brevity. The equality checks $a = b$ in algorithms are performed using $abs(a - b) < 10^{-9}$ in implementation.

3.1 Gaussian σ-truncation

Although the vocabulary size of modern LLMs is on the order of 100k entries, final predictions are typically concentrated within fewer than 100 vocabulary entries per token, regardless of the model used or modifications such as finetuning or quantization. These few high-probability entries form the outlier region of the logit distribution, while the vast majority of the vocabulary forms Gaussian

[1] A Triton Program is equivalent to a CUDA Threadblock or a ROCm Workgroup.

Algorithm 2 Quaternary Top-k Pivot Search

Input: Search set $\mathbf{Z_s}$, Top-k target k
Output: Pivot $\tau,\ z_{dup},\ N_{dup}$
$\tau \leftarrow \infty,\ z_{dup} \leftarrow 0,\ N_{dup} \leftarrow 0$ // Initialize output values
$l \leftarrow \min(Z_s),\ r \leftarrow \max(Z_s),\ i \leftarrow 0$ // Initialize search range
while $(|r - l| \geq 10^{-9}) \ \wedge \ (i < 18)$ **do** // Iterate until range converges or max iterations
  for n in $1, 2, 3$ **do** // Evaluate three pivots (quaternary search)
    $\tau_n \leftarrow l + (n/4) \times (r - l)$ // Set pivots at 25th, 50th, 75th percentile
    $N_{\tau_n} \leftarrow |\{z_i \mid z_i > \tau_n\}|$ // Count elements above pivot
    $z_{mn} \leftarrow \min(\{z_i \mid z_i > \tau_n\})$ // Track minimum logit above pivot
    $N_{z_{mn}} \leftarrow |\{z_i \mid z_i = z_{mn}\}|$ // Count occurrences of minimum logit
    if $(N_{\tau_n} \geq k) \wedge \ (N_{\tau_n} - N_{z_{mn}} < k)$ **then** // Termination: Pivot straddles k with duplicates
      $\tau \leftarrow \tau_n,\ z_{dup} \leftarrow z_{mn},\ N_{dup} \leftarrow N_{z_{mn}}$ // Record pivot and duplicate info
      break
    end if
  end for
  $r \leftarrow (N_{\tau_1} < k)?\ \tau_1 : (N_{\tau_2} < k)?\ \tau_2 : (N_{\tau_3} < k)?\ \tau_3 : r$ // Narrow upper bound
  $l \leftarrow (N_{\tau_3} > k)?\ \tau_3 : (N_{\tau_2} > k)?\ \tau_2 : (N_{\tau_1} > k)?\ \tau_1 : l$ // Narrow lower bound
  $i \leftarrow i + 1$ // Increment iteration
  $\tau \leftarrow (l + r)/2$ // Set fallback pivot to midpoint
end while
return $(\tau,\ z_{dup},\ N_{dup})$

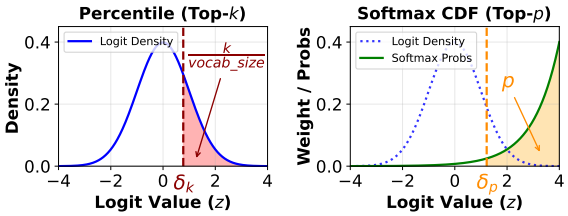

Figure 1: Gaussian σ-truncation on logits following a standard normal distribution.

noise [21]. The goal of Gaussian-based σ-truncation is to truncate this massive noise region in a **single linear pass**, while its large headroom of a few hundred vocabulary entries allows reliable capturing and forwarding of the real predictions to the quaternary search for exact Top-k and Top-p sampling. This step limits the **complexity of Qrita to** $\mathcal{O}(n)$, providing a much more efficient bound over the ever-growing LLM vocabulary compared to existing $\mathcal{O}(n \log(n))$ Top-k and Top-p algorithms.

Let $\mathbf{Z} \in \mathbb{R}^{\mathcal{V}}$ denote the model logits. We first approximate the distribution of logits as a quasi-Gaussian $\mathbf{Z} \sim \mathcal{N}(\mu, \sigma^2)$. We sample a block of logits and compute the sample mean μ and standard deviation σ of the logits to determine a truncation threshold $t = \mu + \delta \cdot \sigma$. We use a lookup table to quickly approximate the coefficient δ from the target k or p value. We precompute table entries using the normal quantile function Φ_k^{-1} where $\delta_k = \Phi_k^{-1}(1 - k/\mathcal{V})$ for k and the inverse cumulative softmax probability of the samples Φ_p^{-1} where $\delta_p = \Phi_p^{-1}(p)$.

Figure 1 illustrates Gaussian σ-truncation on logits following a standard normal distribution ($t = \delta$). In practice, we subtract $0.2 \times abs(t)$ from t as a safety margin and gather the truncated *outlier set* that satisfies $\mathbf{Z}_t = \{z_i \mid z_i > t\}$. We allocate additional memory for $\mathcal{V}$ elements per Triton program to store the truncated set. If $|\mathbf{Z}_t| > k$, or a "truncation hit," we pass the pointer for $\mathbf{Z}_t$ to the Quaternary Search; if not, we fall back and pass the pointer for $\mathbf{Z}$. For Top-p, we compute the softmax of the logits in this stage.[2] We gather and sum the *outlier probabilities*, and check if the *sum* is larger than p instead. We provide details on the benefits of Gaussian σ-truncation and its hit-rate in Section 4.3.

3.2 Quaternary Pivot Search

Even with the Gaussian σ-truncation, the overhead of pivot search is still significant. To mitigate this, we use the Quaternary Pivot Search in Algorithm 2. We increase the number of pivots to three to halve the number of iterations and track the occurrence of the smallest logit that is larger than the pivot.

[2]Softmax on the unsorted logits may produce different output to softmax after sorting due to floating point non-associativity.

Algorithm 3 Top-k output with Duplication Handling

1: **Input:** Logits set $\mathbf{Z}$, τ, z_{dup}, N_{dup}
2: **Output:** Output $\mathbf{O}$
3: $\mathbf{M}_\tau \leftarrow 1(\mathbf{Z} > \tau)$ // Base mask: all logits above pivot
4: $\mathbf{N}_\tau \leftarrow \Sigma M_\tau$ // Number of all logits above pivot
5: $N_{keep} \leftarrow N_{dup} - (N_\tau - k)$ // Number of duplicate logits to retain
6: $\mathbf{M}_{z_{dup}} \leftarrow 1(\mathbf{Z} = z_{dup})$ // Locate all occurrences of the duplicate boundary logit
7: $\mathbf{C}_{z_{dup}} \leftarrow cumsum(\mathbf{M}_{z_{dup}}) \wedge M_{z_{dup}}$ // Occurrence map: Index duplicates in appearance order
8: $\mathbf{M}_{remv} \leftarrow 1(\mathbf{C}_{z_{dup}} > N_{keep})$ // Mark excess duplicates for removal
9: $\mathbf{M}_{final} \leftarrow \mathbf{M}_\tau \wedge \neg\mathbf{M}_{remv}$ // Final mask: Base set minus removed duplicates
10: $\mathbf{O} \leftarrow \mathbf{Z} + (1 - \mathbf{M}_{final}) \cdot (-\infty)$ // Apply mask in-place; masked logits set to $-\infty$
11: **return** $\mathbf{O}$

This prevents the search from falling into an excessive number of iterations and nondeterministic behavior in the presence of duplicate logits, to be explained later.

We first set our initial search range $r \leftarrow \max(\mathbf{Z}_s)$ and $l \leftarrow \min(\mathbf{Z}_s)$ (line 4), where $\max(\mathbf{Z}_s)$ is $\max(\mathbf{Z})$ and $\min(\mathbf{Z}_s)$ is t if the truncation hits and $\min(\mathbf{Z})$ if not. For each iteration, we set the search pivots τ_n ($n = 1, 2, 3$) to the 25th, 50th, 75th percentiles of the search range (line 7) and count $N_{\tau_n} = |\{z_i \mid z_i > \tau_n\}|$ (line 8). We also track the *minimum logit* z_{mn} that is larger than τ_n, or $z_{mn} = \min(\{z_i \mid z_i > \tau_n\})$ (line 9), and its occurrence $N_{z_{mn}} = |\{z_i \mid z_i = z_{mn}\}|$ (line 10).

The search ends when $(N_{\tau_n} \geq k) \wedge (N_{\tau_n} - N_{z_{mn}} < k)$ (line 11), which finds the pivot τ_n where the exclusion of Z_{mn} logits pushes N_{τ_n} below k. This provides a robust termination condition for the pivot search, as it detects the case where multiple Z_{mn} logits exist and must be partially discarded to satisfy Top-k. Without this, the naive $N_{\tau_n} = k$ condition will fall into an infinite loop. Once this condition is met, we return τ_n, z_{mn}, and $N_{z_{mn}}$, now labeled as τ, z_{dup}, and N_{dup}. If the condition is not met for any of the pivots, we update the search range. If $N_{\tau_1} < k$, this means that the true $\tau < \tau_1$, therefore $r \leftarrow \tau_1$. Similarly, if $N_{\tau_3} > k$, $\tau > \tau_3$, therefore $l \leftarrow \tau_3$. We chain the update as in lines 16 and 17 of Algorithm 2, and iterate until the search range $|r - l|$ falls below the tolerance of 10^{-9}, or up to 18 iterations, which reduces the range to $1/4^{18} = 1/2^{36}$ of the original.

For Top-p, we use the same algorithm but with pivot probability π_n, and calculate the sum of probabilities $S_{\pi_n} = \sum_{p_i > \pi_n} p_i$ while tracking $p_{mn} = \min(\{p_i \mid p_i > \pi_n\})$ and $S_{p_{mn}} = \sum_{p_i = p_{mn}} p_i$. The search terminates when $(S_{\pi_n} \geq p) \wedge (S_{\pi_n} - S_{p_{mn}} < p)$, and we still count $N_{p_{mn}}$ to be later used for duplication handling. For range updates, we do $r \leftarrow (S_{\pi_1} < p)?\pi_1 : (S_{\pi_2} < p)?\pi_2 : (S_{\pi_3} < p)?\pi_3 : r$ and $l \leftarrow (S_{\pi_3} > p)?\pi_3 : (S_{\pi_2} > p)?\pi_2 : (S_{\pi_1} > p)?\pi_1 : l$.

We also experiment with a larger number of pivots. However, due to the increased pressure on hardware resources such as the register file, the per-iteration time increases significantly to a point where it outweighs the benefits from reduced iterations. We provide details on the tradeoff between binary and quaternary search in Section 4.5.

3.3 Duplication Handling

Algorithm 3 shows the final output generation with duplication handling. First, we create a mask $\mathbf{M}_\tau = 1(\mathbf{Z} > \tau)$ (line 3), as our base set output logits. As $\mathbf{M}_\tau$ contains all duplicate logits, we must partially zero out the duplicate logits. We calculate the number of duplicate logits that we keep using $N_{keep} = N_{dup} - (N_\tau - k)$ for Top-k, and $N_{keep} = N_{dup} - \lfloor (S_\tau - p)/p_{mn} \rfloor$ for Top-p (line 5). To efficiently locate and remove unwanted duplicate logits in parallel, we first create a mask of duplicate logits $\mathbf{M}_{z_{dup}}$ (line 6) and use cumulative sum on the mask and perform logical AND with the original mask to create the *occurrence map* of the duplicate logits, $\mathbf{C}_{z_{dup}}$ (line 7). We then create a mask of duplicate logits we remove, $\mathbf{M}_{remv}$, by comparing the occurrence map $\mathbf{C}_{z_{dup}}$ with N_{keep} (line 8). Then we perform logical AND between $\mathbf{M}_\tau$ and $\neg\mathbf{M}_{remv}$ to get our final mask, $\mathbf{M}_{final}$ (line 9).

$\mathbf{M}_{final}$ maps whether each logit of Z is included in our final Top-k or Top-p output. It can be used to gather the final output or mask out unwanted logits. As we support execution with different k or p values per batch dimension, we mask out unwanted logits in-place from Z (line 10), and return our final output O. As we calculate the exact number of duplicate logits that must be kept, and always keep the ones that appear earlier in the logits set, our algorithm keeps the correct semantics of stable sort-based Top-k and Top-p algorithms with reproducible and deterministic outputs across all runs.

3.4 Tiered Triton Block Size Autotune

We also include a Triton-specific optimization in our implementation. The vectorized operators of our algorithm, such as $\min()$, $\max()$, masking, or $cumsum()$, are executed using *Triton blocks*. Triton splits the vector into fixed-size blocks, and the block elements are executed in parallel across GPU threads. If the blocks are too small, multiple iterations are needed to cover the whole vector, and if the blocks are too large, the blocks are padded, reducing hardware utilization. However, in Qrita, the size of the vector operands differs greatly depending on whether the Gaussian σ-truncation is a hit or not. Therefore, we create tiered block sizes, where the larger block size is used for operations on the full logit set, and the smaller block size is used for operations on the truncated set. We use Triton Autotune to let the compiler find the best combination of block sizes. We show the benefit of our multi-block autotune in Section 4.5.

4 Evaluation

4.1 Evaluation Setup

We evaluate Qrita by comparing its performance against four popular baselines: a naive PyTorch [17] sort-based Top-k and Top-p implementation, vLLM [12][3], FlashInfer [27], and SGLang [28]. Note that FlashInfer and SGLang return a single output token, which is stochastically equivalent to sampling from Top-k or Top-p. As such, they have a much more relaxed output constraint and are not equivalent to the strict definition of Top-k or Top-p. However, we still include them as baselines as they are the current state-of-the-art solutions for Top-k and Top-p sampling. We use samples from the WikiText-2 [14] dataset to generate the input logits using five different LLMs, R1-Distill-Llama-8B [6, 4], Qwen3-8B [26], GPT-OSS-20B [1], Qwen3.5-35B-A3B [18], and Gemma3-4B [11], which have vocabulary sizes of 128256, 151936, 201088, 248320, and 262208 respectively. All experiments are executed on an NVIDIA H100 GPU, unless stated otherwise.

4.2 Performance

We present the execution throughput of Qrita and the baselines on NVIDIA H100 in Figure 2. We execute a batched logits tensor of shape [*Batch_size*, *Vocab_size*], and divide the batch size by the execution time to get our throughput measurement. Overall, Qrita outperforms all baselines in most configurations, with a few exceptions where SGLang or FlashInfer, which have relaxed output constraints with reduced functionality, achieve comparable throughput. As such, Qrita can be considered as a direct improvement to the baselines in both performance and functionality. Interestingly, the throughput of Qrita scales linearly as batch size increases up to 128, and the throughput increase slows down after. This is because we allocate a Triton Program per multiprocessor of the GPU, of which the H100 has 132. As such, our current Qrita implementation has a scalability limitation in the batch dimension. This can be solved by creating a Qrita implementation where each Triton Program handles multiple batches in parallel, which we leave as future work.

Figure 3 shows the throughput results on RTX4090 and AMD MI350X. For RTX4090, Qrita achieves up to 2$\times$ the throughput of baselines even when compared to SGLang or FlashInfer. Considering that Top-k and Top-p are mostly memory-bound operations, we attribute this difference to the memory improvements of Qrita through the use of search space truncation and the sorting-free search, performing favorably on RTX4090 with reduced memory performance. On MI350X, Qrita is unmatched by all baselines, as Qrita outclasses PyTorch and vLLM while the high-performance sampler implementations of FlashInfer and SGLang are available only on NVIDIA GPUs.

4.3 Truncation Hit-rate and Memory Use

Table 1 presents the hit-rate of the Gaussian σ-truncation, along with the average number of outliers and average sum of outlier probabilities of the truncated sets while running the Wikitext-2 dataset. We observe that the current truncation threshold for Top-k is tuned to be extremely conservative, achieving a constant hit-rate of 1.0 and a headroom of hundreds of logits even though k is set to only 50. It is possible to gain speedup by tightening the σ-truncation threshold, but we find the benefits marginal except for the Gemma3 case. This is because modern GPUs have enough thread resources

[3]We compare to vLLM's default sampler before Qrita.

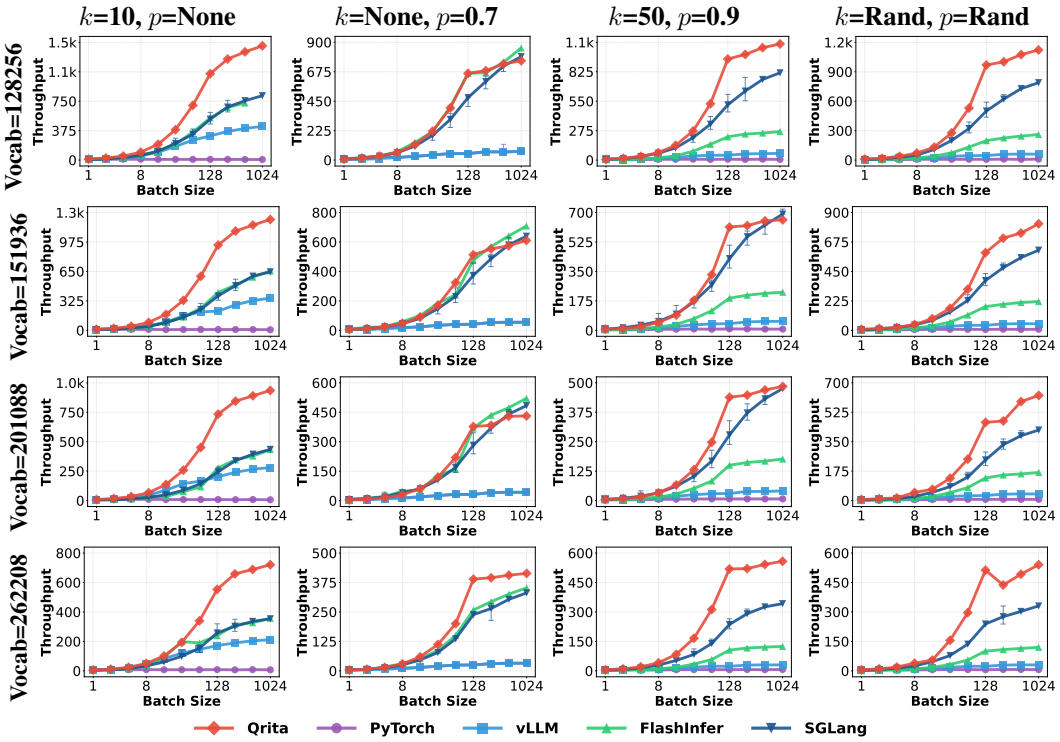

Figure 2: Sampling throughput (samples/ms) over various models and executions on an NVIDIA H100, averaged across 100 runs with a confidence interval of 96%.

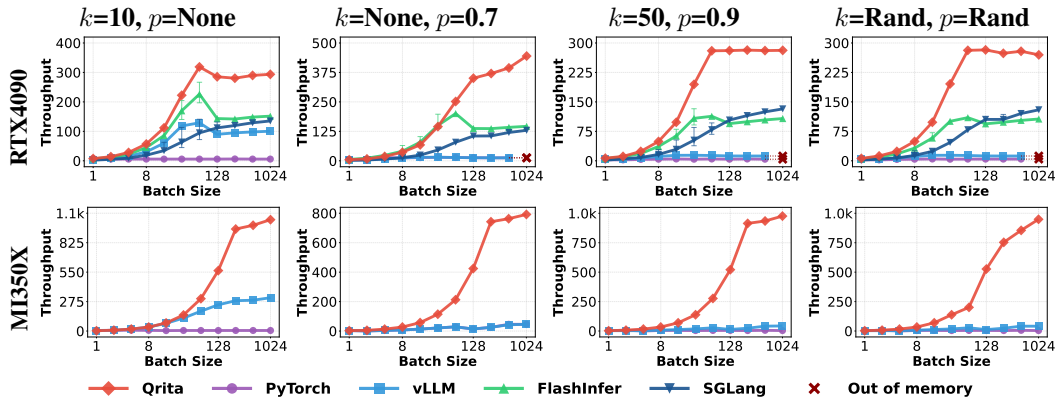

Figure 3: Sampling throughput (samples/ms) of Gemma3-4B over various executions on NVIDIA RTX4090 and AMD MI350X, averaged across 100 runs with a confidence interval of 96%. High-performance samplers of FlashInfer and SGLang are written in CUDA and are unavailable on AMD.

to handle hundreds of logits in parallel, with minimal impact on latency. However, if we extend Qrita to support multiple batches per Triton program as mentioned before, it is likely that a tighter threshold on each batch will be beneficial for efficient thread-level parallelism across the batch dimension.

For the Top-p case, our results show that while our Gaussian-based modeling of the logits provides robust lower-bound truncation thresholds, larger variances in the sum of probability and hit-rate exist across models compared to the Top-k case. This is because the inclusion of outlier values in the real logit distribution greatly alters the probability calculation of the softmax operator compared to pure Gaussian-based modeling. As such, to achieve a more accurate truncation, it may be beneficial to include a profiling step that creates a dedicated δ lookup table for each model.

Figure 4 presents the memory usage of Qrita and baselines across different executions on a logarithmic scale. While Qrita achieves lower memory usage compared to all baselines, it can also be implemented in-place to have *zero memory use* by skipping Gaussian σ-truncation. However, we believe that the

Model	$k = 50$		$p = 0.9$	
	N_t	Hitrate	S_t	Hitrate
R1-Llama-8B	276	1.0	0.984	0.985
Qwen3-8B	399	1.0	0.967	0.901
GPT-OSS-20B	286	1.0	0.952	0.938
Gemma3-4B	2368	1.0	0.999	1.0

Table 1: Gaussian σ-truncation hitrate with avg. number of outliers N_t and avg. sum of outlier probabilities S_t.

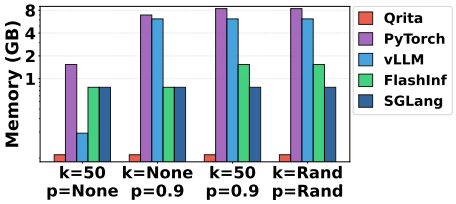

Figure 4: Memory usage during sampling. Vocab size=201088, Batch size=1024

performance benefit from truncation is large enough to justify a buffer space, as shown in ablation studies. Alternatively, as the truncated set size is often a few thousand elements at most, the truncation buffer may be reduced from the current full vocabulary buffer to further optimize Qrita memory use.

4.4 End-to-end Performance

Model	Input/Output Tokens	Sampling Skipped	k=50		p=0.9		k=50, p=0.9		Average Speedup
			Baseline	Qrita	Baseline	Qrita	Baseline	Qrita	
R1-Dist-Llama-8B	256/1024	14.80	12.70	14.50	12.49	14.22	12.39	14.54	1.15×
R1-Dist-Llama-8B	1024/256	26.28	24.56	26.02	24.36	25.63	24.29	26.00	1.06×
R1-Dist-Llama-8B	1024/1024	14.21	12.78	14.03	12.61	13.76	12.56	14.03	1.10×
Qwen3-8B	256/1024	13.95	12.04	13.68	11.79	13.57	11.57	13.66	1.16×
Qwen3-8B	1024/256	25.75	23.22	25.50	23.08	25.23	22.95	25.40	1.10×
Qwen3-8B	1024/1024	13.29	11.83	13.09	11.67	12.98	11.62	13.11	1.12×
GPT-OSS-20B	256/1024	30.08	20.15	28.39	19.66	26.43	19.32	28.55	1.41×
GPT-OSS-20B	1024/256	52.13	43.85	51.72	43.31	47.72	42.63	52.10	1.17×
GPT-OSS-20B	1024/1024	35.16	25.84	33.50	25.21	31.32	24.73	33.67	1.30×
Gemma3-4B	256/1024	15.63	12.01	15.41	11.67	15.27	11.51	15.53	1.31×
Gemma3-4B	1024/256	35.45	29.28	34.09	28.72	33.85	28.49	34.11	1.18×
Gemma3-4B	1024/1024	19.29	14.83	18.28	14.48	18.21	14.30	18.42	1.26×

Table 2: End-to-end serving throughput of baseline vLLM, vLLM with sampling kernel skipped (theoretical max performance), and vLLM with Qrita on NVIDIA H100 (KTokens/sec).

Model	k	p	Impl	TTFT (ms)			TPOT (ms)			E2EL (ms)		
				p50	p95	p99	p50	p95	p99	p50	p95	p99
Qwen3.5-35B-A3B	50	0.9	Baseline	77.24	114.55	145.80	21.90	23.51	24.20	5512.51	6064.16	6090.64
Qwen3.5-35B-A3B	50	0.9	Qrita	71.86	123.28	132.75	19.28	20.51	20.87	4882.88	5288.63	5322.34
Qwen3.5-35B-A3B	50	none	Baseline	78.10	114.45	134.24	22.05	23.56	23.97	5491.71	6068.22	6086.06
Qwen3.5-35B-A3B	50	none	Qrita	77.15	136.00	163.68	20.56	22.25	24.63	5196.33	5727.92	5739.14
Qwen3.5-35B-A3B	none	0.9	Baseline	75.96	114.73	140.62	22.09	23.64	23.73	5631.94	6106.15	6125.16
Qwen3.5-35B-A3B	none	0.9	Qrita	75.95	147.27	179.28	20.63	22.11	22.27	5254.81	5705.91	5738.00

Table 3: End-to-end serving latency of baseline vLLM and vLLM with Qrita on NVIDIA H100.

Table 2 reports the end-to-end serving performance improvement of Qrita by comparing the serving throughput of baseline vLLM execution and vLLM with Qrita as its Top-k and Top-p sampler. We also report the theoretical max throughput when the Top-k and Top-p sampling kernel is entirely skipped. As can be seen in the table, Qrita consistently outperforms the vLLM baseline, delivering up to 1.41× speedup and often **providing performance within 5% of the theoretical max values**.

We note that the speedup is affected by the ratio between active model parameters and vocabulary size. For example, GPT-OSS-20B only has 3B active parameters, being an MoE model, and therefore achieves the best speedup among the tested models. Similarly, Qwen3-Next-80B-A3B [26] with 3B active parameters may be expected to have roughly 2.7× the performance improvement compared to Qwen3-8B, or up to 40% end-to-end throughput improvement similar to GPT-OSS-20B. Therefore, while the end-to-end benefits of Qrita are largely model-dependent, it can *significantly improve the inference performance* of small to mid-sized dense LLMs, or parameter-efficient MoE models.

Comparing the Qrita and vLLM's time-to-first-token (TTFT), time-per-output-token (TPOT), and end-to-end latency (E2EL) results in Table 3, we see that there are cases where Qrita delivers a slower p95/p99 tail latency for TTFT and TPOT. However, Qrita consistently delivers lower average (p50) latencies, resulting in around an 5-13% reduction in end-to-end latencies even in p95/p99 cases.

	Trunc. Hit	Trunc. Fallback	No Trunc.	Quaternary	Binary	Sort	Dup.	Autotune	Time (ms)
Run A	O	X	X	O	X	X	O	O	0.919
Run B	O	X	X	O	X	X	O	X	1.354
Run C	O	X	X	O	X	X	X	X	1.308
Run D	X	O	X	O	X	X	X	X	5.373
Run E	X	X	O	O	X	X	X	X	4.931
Run F	O	X	X	X	O	X	X	X	1.281
Run G	X	O	X	X	O	X	X	X	7.773
Run H	X	X	O	X	O	X	X	X	7.274
Run I	O	X	X	X	X	O	X	X	2.099
Run J	X	X	O	X	X	O	X	X	16.487

Table 4: Ablation study of Qrita techniques. k=50, p=0.9, Vocab size=128256, Batch size=1024

4.5 Ablation Studies

In Table 4, we provide ablation studies on the various techniques of Qrita. Run A is the full Qrita execution, which includes all key techniques introduced in Section 3. We first study the effect of the **Tiered Triton Block Size Autotune** by comparing Run A with Run B, which uses only a single block size of 4096 for all vectorized operators. Autotuning reduces the execution latency by almost 30%, proving to be a critical optimization for Qrita that operates on vectors of varying sizes.

The **overhead of duplication handling** is shown by comparing Run B with Run C, which excludes the duplicate logit removal logic from the output mask generation. Latency is slightly improved by about 3.5%. We include duplication handling for the sake of correctness, but if performance is critical and a more relaxed output constraint can be tolerated, duplication handling can be removed.

Runs C, D, and E compare **the effect of Gaussian σ-truncation** by comparing the time taken when the truncation hits, when it misses, and when it is excluded. The truncation delivers the biggest speedup of all techniques used, providing 74% latency reduction compared to the excluded case that must search the full logits set. By comparing Runs D and E, we also observe that the overhead of the truncation is about 0.4ms, which is quite significant when compared to fully optimized Run A, contributing 48% of the total runtime. As such, reducing the overhead of Gaussian σ-truncation may be a good future direction for Qrita.

By comparing Runs F, G, and H against Runs C, D, and E, we observe **the impact of quaternary search over binary search** for pivot finding. Surprisingly, quaternary search is about 2% slower than binary search for the truncation hit case, as the reduced search space minimizes the impact of search iterations. However, due to the large absolute latency reduction of 2.3ms it provides in the fallback case, we keep quaternary search as our default approach.

Comparing Runs I and J against Runs C and E, we also observe **the benefit of quaternary search over the sorting-based algorithms**, where it provides about 38% and 70% reduction in latency, proving the benefit of the more memory-efficient search-based algorithm over the sorting-based algorithm. Note that the configuration of Run J is identical to the naive PyTorch baseline.

4.6 Empirical Exactness

We use the Top-k and Top-p sampler test suite of vLLM[4] to compare Qrita against the PyTorch sort-based reference. The tests sweep across different batch sizes, vocabulary sizes, and k and p values, including requests where Top-k or Top-p is skipped for part of the batch. They also cover edge cases: extreme k ($k = 1$ and $k = \mathcal{V}$), extreme p (p near 0 and $p = 1$), identical logits, near-identical logits (spread of 10^{-6}), and $-\infty$ logits (e.g., from structured output masks), including checks that no NaN is produced. Qrita passes all tests, giving a match rate of 100% and a miss rate of 0% against the PyTorch reference on this suite.

5 Discussions

Comparison to Radix Top-k, Radix Top-p, and other GPU Top-k and Top-p Algorithms: Qrita is composed of three distinct components serving different purposes. The component comparable to Radix Top-k and Top-p or other GPU Top-k and Top-p algorithms is the Quaternary Pivot Search, as it

[4] https://github.com/vllm-project/vllm/blob/main/tests/v1/sample/test_topk_topp_sampler.py

is the main "exact Top-k and Top-p" algorithm. On the other hand, the goal of Gaussian σ-truncation is *to reduce the Top-k and Top-p search space in a single pass*, and the goal of Duplication Handling is to *ensure deterministic results* in the presence of identical values, which is not directly comparable to other GPU Top-k and Top-p algorithms, as they add on top of an exact Top-k and Top-p algorithm.

We chose the search-based approach for the "exact Top-k and Top-p" component, as it is the easiest to fold the detection and counting of duplicate logits into the algorithm. Radix Top-k and Top-p, RadiK, and many high-performance GPU Top-k and Top-p algorithms are inefficient or non-deterministic in the presence of duplicate values compared to our Quaternary Search, which can detect the pivot falling on duplicate values. Our choice of Quaternary Search is based on the effectiveness it brings to duplicate handling, and not because it is necessarily better than radix-based algorithms in isolation.

Computational Complexity of Qrita: The computational complexity of the Gaussian σ-truncation is $\mathcal{O}(N)$ as it passes twice over the logit set of size N. If the truncated set size is M, the complexity of quaternary search is $\mathcal{O}(M \log(M))$, as we need $\log(M)$ passes on the logit set of size M. As $M \ll N$ (k or $p \ll vocab_size$), the complexity of Qrita is $\mathcal{O}(N)$. However, RadixSelect or Quickselect may outperform quaternary search with large M, as they also achieve $\mathcal{O}(M)$ by reducing the search space using gather on each step. We do not apply such optimization because (1) we limit M using σ-truncation, and (2) gathering operations on GPUs are very inefficient. Current quaternary search performs scan operations using parallel threads, which are mapped to contiguous reads. However, gather operations require random read and write operations, which limit batching of memory accesses and add the overhead of write accesses, causing higher latency per step. This is why we limit the gather, or the search space reduction, to only the Gaussian σ-truncation pass.

Exactness Analysis: Despite the empirical correctness shown using a test suite in Section 4.6, it is possible that Qrita may mismatch some PyTorch sort-based Top-k and Top-p references in theory.

(a) Gaussian σ-truncation. We gather outlier logits strictly larger than the threshold t. Therefore, the outlier set of size M must contain the top-M values of the logit set. If $M \geq k$, the outlier set is a superset of the Top-k set. The same holds for Top-p if the sum of the outlier probabilities is larger than p. If the outlier set does not satisfy this property, we fall back to the full logit set. Thus, the truncated set is guaranteed to be a superset of the Top-k or Top-p set, and Qrita is correct if the remaining search finds the exact set within it.

(b) Quaternary pivot search. The search is correct if its resolution, or its ability to differentiate between two adjacent values, is finer than the gap between the two adjacent values of the pivot.

(b-1) Absolute resolution. The absolute resolution is the tolerance of 10^{-9} (roughly 2^{-30}). Logits near the cutoff pivot that differ by less than 10^{-9} may be treated as identical (duplicate) values. However, this is unlikely in practice, as logits are typically FP16 or BF16 values upcast to FP32. The finest resolution of FP16 is $2^{-24} \approx 6.0 \times 10^{-8}$, which is coarser than 10^{-9}. For BF16, with its 7-bit mantissa, a resolution finer than 10^{-9} only occurs for $|z| < 2^{-22} \approx 2.4 \times 10^{-7}$. With reasonable k and p values and real model predictions, pivots rarely land in such near-zero values.

(b-2) Relative resolution. Since we reduce the search range by $4\times$ per iteration for up to 18 iterations, the relative resolution is $4^{-18} \approx 1.5 \times 10^{-11}$ times the original search range. Logits near the pivot that differ by less than this may lead to an incorrect pivot. For probabilities, which lie within $[0, 1]$, the relative resolution is at most 1.5×10^{-11}. For logits, it is finer than the absolute resolution of 10^{-9} whenever the search range is below about 68, which is the case for typical logit distributions.

(c) Duplication handling. We keep the first-occurrence set, which is *equivalent to a stable sort-based Top-k and Top-p* that preserves the order of identical elements. However, **PyTorch's sort is unstable by default**, with undefined order between identical elements, unless the `stable` flag is set. Therefore, Qrita results may differ from PyTorch's default sort. However, this is not necessarily incorrect, and we believe that equivalence to a stable sort is the most reasonable design choice for determinism.

6 Conclusion

This paper proposes Qrita, an efficient Top-k and Top-p truncation algorithm for GPUs. Our method supports both Top-k and Top-p in $\mathcal{O}(n)$ while preserving the precise output semantics of both algorithms. We implement Qrita in Triton and show that Qrita achieves up to $1.4\times$ higher throughput against state-of-the-art baselines, providing a high-performance solution for LLM decoding.

Acknowledgments and Disclosure of Funding

We would like to thank Woosuk Kwon, Yilong Zhao, Mayank Mishra, the vLLM team, and the UC Berkeley Sky Computing Lab for sharing their valuable insights in this research.

This research was supported by the Amazon AI PhD Fellowship awarded to Jongseok Park, as well as NSF awards IIS-1955488, IIS-2027575, DOE awards DE-SC0016260, AC02-05CH11231, DARPA Agreement No. HR00112590131, and gifts to the UC Berkeley Sky Computing Lab from Accenture, Algorithmic SuperIntelligence Labs, Amazon, AMD, Anyscale, Broadcom, cmpnd, Google, IBM, Intel, Intesa Sanpaolo, Lambda, Lightspeed, Mirendil, NVIDIA, Soma Labs, and VESSL.

References

[1] Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sebastien Bubeck, Che Chang, Kai Chen, Mark Chen, Enoch Cheung, Aidan Clark, Dan Cook, Marat Dukhan, Casey Dvorak, Kevin Fives, Vlad Fomenko, Timur Garipov, Kristian Georgiev, Mia Glaese, Tarun Gogineni, Adam Goucher, Lukas Gross, Katia Gil Guzman, John Hallman, Jackie Hehir, Johannes Heidecke, Alec Helyar, Haitang Hu, Romain Huet, Jacob Huh, Saachi Jain, Zach Johnson, Chris Koch, Irina Kofman, Dominik Kundel, Jason Kwon, Volodymyr Kyrylov, Elaine Ya Le, Guillaume Leclerc, James Park Lennon, Scott Lessans, Mario Lezcano-Casado, Yuanzhi Li, Zhuohan Li, Ji Lin, Jordan Liss, Lily, Liu, Jiancheng Liu, Kevin Lu, Chris Lu, Zoran Martinovic, Lindsay McCallum, Josh McGrath, Scott McKinney, Aidan McLaughlin, Song Mei, Steve Mostovoy, Tong Mu, Gideon Myles, Alexander Neitz, Alex Nichol, Jakub Pachocki, Alex Paino, Dana Palmie, Ashley Pantuliano, Giambattista Parascandolo, Jongsoo Park, Leher Pathak, Carolina Paz, Ludovic Peran, Dmitry Pimenov, Michelle Pokrass, Elizabeth Proehl, Huida Qiu, Gaby Raila, Filippo Raso, Hongyu Ren, Kimmy Richardson, David Robinson, Bob Rotsted, Hadi Salman, Suvansh Sanjeev, Max Schwarzer, D. Sculley, Harshit Sikchi, Kendal Simon, Karan Singhal, Yang Song, Dane Stuckey, Zhiqing Sun, Philippe Tillet, Sam Toizer, Foivos Tsimpourlas, Nikhil Vyas, Eric Wallace, Xin Wang, Miles Wang, Olivia Watkins, Kevin Weil, Amy Wendling, Kevin Whinnery, Cedric Whitney, Hannah Wong, Lin Yang, Yu Yang, Michihiro Yasunaga, Kristen Ying, Wojciech Zaremba, Wenting Zhan, Cyril Zhang, Brian Zhang, Eddie Zhang, and Shengjia Zhao. gpt-oss-120b & gpt-oss-20b model card, 2025.

[2] 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, 2012.

[3] Fu Bang. Gptcache: An open-source semantic cache for llm applications enabling faster answers and cost savings. In *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)*, pages 212–218, 2023.

[4] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[5] Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation, 2018.

[6] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan

Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat,

Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024.

[7] Horace He and Thinking Machines Lab. Defeating nondeterminism in llm inference. *Thinking Machines Lab: Connectionism*, 2025. https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/.

[8] John Hewitt, Christopher D. Manning, and Percy Liang. Truncation sampling as language model desmoothing, 2022.

[9] Charles AR Hoare. Algorithm 65: find. *Communications of the ACM*, 4(7):321–322, 1961.

[10] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration, 2020.

[11] Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry, Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi, Dan Malkin, Ravin Kumar, David Vilar, Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa Saade, Alex Feng, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, András György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia Paterson, Ashish Shenoy, Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini, Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivakumar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eugene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna Klimczak-Plucińska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian Ballantyne, Idan Szpektor, Ivan Nardini, Jean Pouget-Abadie, Jetha Chan, Joe Stanton, John Wieting, Jonathan Lai, Jordi Orbay, Joseph Fernandez, Josh Newlan, Ju yeong Ji, Jyotinder Singh, Kat Black, Kathy Yu, Kevin Hui, Kiran Vodrahalli, Klaus Greff, Linhai Qiu, Marcella Valentine, Marina Coelho, Marvin Ritter, Matt Hoffman, Matthew Watson, Mayank Chaturvedi, Michael Moynihan, Min Ma, Nabila Babar, Natasha Noy, Nathan Byrd, Nick Roy, Nikola Momchev, Nilay Chauhan, Noveen Sachdeva, Oskar Bunyan, Pankil Botarda, Paul Caron, Paul Kishan Rubenstein, Phil Culliton, Philipp Schmid, Pier Giuseppe Sessa, Pingmei Xu, Piotr Stanczyk, Pouya Tafti, Rakesh Shivanna, Renjie Wu, Renke Pan, Reza Rokni, Rob

Willoughby, Rohith Vallu, Ryan Mullins, Sammy Jerome, Sara Smoot, Sertan Girgin, Shariq Iqbal, Shashir Reddy, Shruti Sheth, Siim Põder, Sijal Bhatnagar, Sindhu Raghuram Panyam, Sivan Eiger, Susan Zhang, Tianqi Liu, Trevor Yacovone, Tyler Liechty, Uday Kalra, Utku Evci, Vedant Misra, Vincent Roseberry, Vlad Feinberg, Vlad Kolesnikov, Woohyun Han, Woosuk Kwon, Xi Chen, Yinlam Chow, Yuvein Zhu, Zichuan Wei, Zoltan Egyed, Victor Cotruta, Minh Giang, Phoebe Kirk, Anand Rao, Kat Black, Nabila Babar, Jessica Lo, Erica Moreira, Luiz Gustavo Martins, Omar Sanseviero, Lucas Gonzalez, Zach Gleicher, Tris Warkentin, Vahab Mirrokni, Evan Senter, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, Yossi Matias, D. Sculley, Slav Petrov, Noah Fiedel, Noam Shazeer, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Jean-Baptiste Alayrac, Rohan Anil, Dmitry, Lepikhin, Sebastian Borgeaud, Olivier Bachem, Armand Joulin, Alek Andreev, Cassidy Hardin, Robert Dadashi, and Léonard Hussenot. Gemma 3 technical report, 2025.

[12] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023.

[13] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023.

[14] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

[15] Prabhat Nagarajan, Garrett Warnell, and Peter Stone. Deterministic implementations for reproducibility in deep reinforcement learning, 2019.

[16] Minh Nhat Nguyen, Andrew Baker, Clement Neo, Allen Roush, Andreas Kirsch, and Ravid Shwartz-Ziv. Turning up the heat: Min-p sampling for creative and coherent llm outputs, 2025.

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[18] Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026.

[19] Nadathur Satish, Mark Harris, and Michael Garland. Designing efficient sorting algorithms for manycore gpus. In *2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–10. IEEE, 2009.

[20] Rylan Schaeffer, Joshua Kazdan, and Yegor Denisov-Blanch. Min-p, max exaggeration: A critical analysis of min-p sampling in language models, 2025.

[21] Chenxia Tang, Jianchun Liu, Hongli Xu, and Liusheng Huang. Top-$n\sigma$: Not all logits are you need, 2024.

[22] Philippe Tillet. *Triton Language API Documentation*, 2020. Accessed: 2026-05-06.

[23] Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 10–19, 2019.

[24] Xi Xie, Yuebo Luo, Hongwu Peng, and Caiwen Ding. Rtop-k: Ultra-fast row-wise top-k selection for neural network acceleration on gpus, 2025.

[25] Shanli Xing, Zihao Ye, Bohan Hou, Luis Ceze, and Tianqi Chen. Sorting-free gpu kernels for llm sampling. `https://flashinfer.ai/2025/03/10/sampling.html`, mar 2025. FlashInfer Blog.

[26] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025.

[27] Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. Flashinfer: Efficient and customizable attention engine for llm inference serving, 2025.

[28] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.