Optimized Speculative Sampling for GPU Hardware Accelerators

Dominik Wagner[1], Seanie Lee[2], Ilja Baumann[1], Philipp Seeberger[1], Korbinian Riedhammer[1], Tobias Bocklet[1]
[1]Technische Hochschule Nürnberg Georg Simon Ohm, [2]KAIST
dominik.wagner@th-nuernberg.de
lsnfamily02@kaist.ac.kr
{ilja.baumann, philipp.seeberger, korbinian.riedhammer, tobias.bocklet}@th-nuernberg.de

Abstract

In this work, we optimize speculative sampling for parallel hardware accelerators to improve sampling speed. We notice that substantial portions of the intermediate matrices necessary for speculative sampling can be computed concurrently. This allows us to distribute the workload across multiple GPU threads, enabling simultaneous operations on matrix segments within thread blocks. This results in *profiling time improvements* ranging from 6% to 13% relative to the baseline implementation, *without compromising accuracy*. To further accelerate speculative sampling, probability distributions parameterized by softmax are approximated by sigmoid. This approximation approach results in significantly greater relative improvements in profiling time, ranging from 37% to 94%, with a minor decline in accuracy. We conduct extensive experiments on both automatic speech recognition and summarization tasks to validate the effectiveness of our optimization methods.

1 Introduction

Large foundational speech and language models based on autoregressive Transformer (Vaswani et al., 2017) architectures have demonstrated remarkable proficiency across a variety of downstream tasks (Hsu et al., 2021; Radford et al., 2022; Touvron et al., 2023b; Achiam et al., 2024). These models frequently increase in size, consequently requiring more memory and computational resources. However, downstream applications, such as dialogue systems, have strict wall-clock constraints and are often required to generate long sequences (Pope et al., 2023; Chi et al., 2023; Fischer et al., 2024). Due to the sequential token generation in autoregressive decoding, latency increases with both the length of the sequence and the size of the model, resulting in a significant barrier to widespread deployment. On many general-purpose GPU hardware accelerator architectures, the increasing size of models leads to more read and write operations between high-bandwidth memory (HBM) and on-chip shared memory (SRAM) at each decoding step, necessitating more memory bandwidth to meet latency constraints (Pope et al., 2023; Dao et al., 2022; Dao, 2024). Consequently, the speed of autoregressive decoding becomes primarily limited by the available memory bandwidth and not by the number of computations that need to be executed on the dedicated hardware (Shazeer, 2019).

In many cases, however, tokens may be accurately generated by much smaller models that require fewer resources. Motivated by this hypothesis, speculative sampling has been developed to accelerate autoregressive sampling (Stern et al., 2018; Xia et al., 2023; Leviathan et al., 2023; Chen et al., 2023a). Speculative sampling employs a small draft model to generate tokens, which are potential future outputs of a larger target model. These drafted tokens are then verified in parallel by the target model, and only tokens that meet the validation criteria are retained as final outputs to ensure generation accuracy. This approach has been shown to significantly reduce the frequency of time-consuming operations, thereby improving inference latency (Leviathan et al., 2023; Chen et al., 2023a).

In this paper, we focus on optimizing the validation part of speculative sampling to further increase the inference speed. Inspired by recent advances in accelerating computations in the attention mechanism (Dao et al., 2022; Dao, 2024), we explore two faster methods for assessing drafted tokens by leveraging the parallelism capabilities of modern GPUs. We identify that a significant portion of the intermediate matrices required for the sampling process can be computed independently. Exploiting this observation, we distribute the workload across multiple GPU threads and simultaneously compute portions of the intermediate output matrices within thread blocks. This optimization method is faster than the non-optimized baseline implementation and *exact* with regard to the decoding outputs, *i.e.*, it generates the same outputs as the non-optimized method.

To further accelerate speculative sampling, we propose using sigmoid as an element-wise approximation to softmax (Bridle, 1989), which is used to parameterize distributions of target and draft models. Since sigmoid is applied to logits in element-wise fashion, it can be computed in parallel and fused with other sampling-related computations. This enables significant acceleration of the overall process, but results in a small accuracy decline due to the *non-exact* nature of the method.

We evaluate our two optimized algorithms on automatic speech recognition (ASR) and summarization tasks, covering draft model sizes between 166M and 2B parameters and target model sizes between 244M and 13B parameters. The *exact* optimization method reduces profiling time between 6% and 13% relative to the baseline implementation without compromising accuracy. Moreover, the *non-exact* optimization method improves profiling time by 37% to 94%, albeit with a small reduction in accuracy. We summarize our main contributions as follows:[1]

- Implementation of an *exact* and consistently faster variant of speculative sampling optimized for GPU hardware accelerators.
- Exploration of sigmoid as an element-wise approximation to softmax in an even faster but *non-exact* variant of speculative sampling.
- Comprehensive evaluation across multiple tasks, covering a wide range of draft and target model combinations.

2 Related work

Techniques such as quantization (Dettmers et al., 2022; Bondarenko et al., 2023; Stock et al., 2021; Nagel et al., 2021), pruning (Voita et al., 2019; Lagunas et al., 2021; Gromov et al., 2024) and knowledge distillation (Sun et al., 2019; Sanh et al., 2019; Jiao et al., 2020; Hsieh et al., 2023) have proven effective in reducing inference latency with minimal performance impact. However, these approaches often require architectural changes or custom training procedures. Efforts specifically targeting the reduction of memory bandwidth bottlenecks during decoding include methods like multi-query attention, which aims to optimize memory usage per attention layer (Shazeer, 2019), or FlashAttention (Dao et al., 2022; Dao, 2024), which aims to reduce the number of read/write operations between HBM and SRAM on GPUs. Pope et al. (2023), achieve improvements in large-scale inference latency by partitioning models and workload across multiple accelerators combined with various low-level optimizations to improve communication efficiency between devices.

Speculative sampling approaches can be broadly categorized based on how drafting and verification are conducted (Xia et al., 2024a). *Drafting* refers to the efficient prediction of multiple future tokens with a draft model and *verification* refers to the methods used to verify the token sequence with the target model. Some works use specialized draft models (Xia et al., 2023; Zhou et al., 2024). Others employ an existing smaller model from the same series (Chen et al., 2023a; Spector and Re, 2023; Leviathan et al., 2023; Yang et al., 2024) or leverage the target model directly for drafting, *e.g.*, by skipping intermediate layers (Zhang et al., 2024b), using special look-ahead tokens (Monea et al., 2023), or additional modeling heads (Stern et al., 2018; Cai et al., 2024; Zhang et al., 2024a). Verification approaches first supported greedy decoding (Stern et al., 2018; Xia et al., 2023; Zhang et al., 2024b) and were subsequently extended to support other methods such as nucleus sampling (Leviathan et al., 2023; Chen et al., 2023a). Recently, methods to verify multiple draft sequences in parallel have also been explored (Miao et al., 2024; Cai et al., 2024; Spector and Re, 2023).

Several studies have experimented with ReLU and sigmoid as alternatives to softmax in the attention mechanism (Bai et al., 2023b; Shen et al., 2023; Hron et al., 2020; Hua et al., 2022; Li et al., 2022; Wortsman et al., 2023; Ramapuram et al., 2024). They either require training new models from scratch or maintain the computational overhead of gathering information along the full sequence length. Other softmax-related optimizations are tailored to reduce the memory requirement during training by computing only smaller fractions of the full softmax output in the backward pass (Lee and Lee, 2023) or leverage word frequencies in the training data to speed up computation (Grave et al., 2017). Shim et al. (2017) approximate softmax by computing only a fraction of the

[1]The source code of our optimized sampling algorithm is available at `https://github.com/dwgnr/optimized-speculative-sampling`.

full input with singular value decomposition. Other approximation methods are specifically designed for custom hardware such as field-programmable gate arrays (Chen et al., 2023b) and application specific integrated circuits (Geng et al., 2018).

3 Method

3.1 Preliminaries

Speculative sampling. Let $M_p^{\texttt{target}}$ be an autoregressive target model, which induces a categorical distribution distribution $p(x|x_{<i+1})$ over a vocabulary $\mathcal{V} = \{x \in \mathbb{N} : 1 \leq x \leq \texttt{vocab\_size}\}$, given the prefix $x_{<i+1} = (x_1, \ldots, x_i)$. Our goal is to use speculative sampling to accelerate the sampling process of discrete tokens $x \in \mathcal{V}$. This is achieved by approximating the target model with a draft model $M_q^{\texttt{draft}}$, resulting in another categorical distribution $q(x|x_{<i+1})$.

First, given the prefix $(x_1, \ldots, x_{i+c-1})$, $\gamma \in \mathbb{N}_+$ draft tokens are sequentially sampled with $M_q^{\texttt{draft}}$: $x_{i+c} \sim q(x|x_{<i+c})$ for $c = 1, \ldots, \gamma$. The draft tokens are then evaluated using the target model $M_p^{\texttt{target}}$, a process that can be performed in parallel. Each draft token $x_{i+c} \in \mathcal{V}$ is accepted if $r_c \leq \tau_c(x_{i+c})$ for $c = 1, \ldots, \gamma$. The terms r_c and $\tau_c(x_{i+c})$ are computed as follows:

$$\begin{aligned} \tau_c(x_{i+c}) &= \min\left(1, \frac{p(x_{i+c}|x_{<i+c})}{q(x_{i+c}|x_{<i+c})}\right) \\ r_c &\sim U(0,1), \end{aligned} \tag{1}$$

where $U(0,1)$ denotes a uniform distribution between 0 and 1. If x_{i+c} is accepted, the process is repeated for the next token x_{i+c+1} until either a token is rejected or all tokens have been accepted. If x_{i+c} is rejected, a token is resampled from the following adjusted distribution instead:

$$x_{i+c} \sim \texttt{max\_norm}\left(p(x|x_{<i+c}) - q(x|x_{<i+c})\right), \tag{2}$$

where $\texttt{max\_norm}(f(x))$ is given by:

$$\begin{aligned} \texttt{max\_norm}(f(x)) &= \frac{\max(0, f(x))}{\sum_{x' \in \mathcal{V}} \max(0, f(x'))} \\ &= \frac{a(x)}{b} \end{aligned} \tag{3}$$

We denote the numerator of Eq. 3 by $a(x)$ and the denominator by b, as these terms are treated separately in subsequent sections.

The underlying concept of speculative sampling is similar to rejection sampling (Neal, 2003). Intuitively, a new token for the target model $M_p^{\texttt{target}}$ is generated by first sampling from a smaller draft model $M_q^{\texttt{draft}}$, which shares the same support ($\mathcal{V}$) as $M_p^{\texttt{target}}$. The token sampled from $M_q^{\texttt{draft}}$ is then evaluated in parallel with $M_p^{\texttt{target}}$, and its acceptance is determined based on the probability ratio defined in Eq. 1. If the token is rejected, a new one is drawn using the modified distribution in Eq. 2.

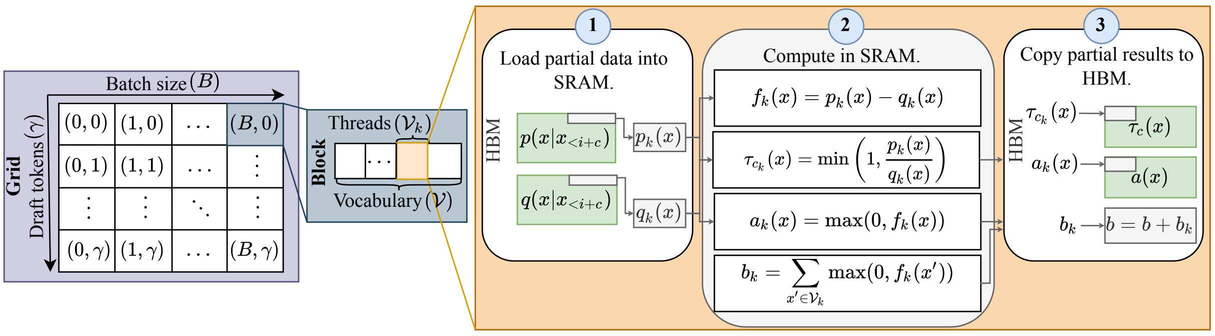
GPU memory and execution model. We briefly describe the memory components and parts of the execution model of GPU hardware accelerators relevant to this work. GPU memory has a hierarchical layout, consisting of various types of memory that differ in size and read/write bandwidth (Jia et al., 2018). Recent GPUs (*e.g.*, NVIDIA's A100 series) typically feature several gigabytes of high-bandwidth memory (HBM) and only a few hundred kilobytes of on-chip shared memory (SRAM) per streaming multiprocessor (SM) (NVIDIA Corporation, 2020). While HBM provides substantial capacity, its memory bandwidth is lower compared to SRAM. The execution model of GPU hardware accelerators involves a large number of threads executing operations known as kernels (Cheng et al., 2014). These threads are organized into thread blocks and assigned to SMs. Each SM partitions its assigned thread blocks into warps of 32 threads, which are then queued for execution on available hardware resources. Each kernel typically follows a pattern: loading inputs from HBM into registers and SRAM, performing computations, and writing the outputs back to HBM.

3.2 Acceleration of speculative sampling

3.2.1 Exact optimization

Our optimization of speculative sampling is designed for parallel heterogeneous hardware accelerators, such as NVIDIA GPUs, which are widely employed to perform inference on large scale models. Similar to the approaches described in Ryoo et al. (2008) and Dao et al. (2022), we aim to redistribute the speculative sampling workload across threads and thread blocks. We load chunks of inputs from HBM to SRAM and make the necessary computations with respect to this input chunk before writing the final result back to HBM.

We notice that the intermediate elements needed for speculative sampling can be computed concurrently within thread blocks and are largely independent of other thread blocks. Specifically, we can compute $(\tau_c(x))_{x \in \mathcal{V}}$ and parts of Eq. 3 in parallel. The kernel is tiled (Lam et al., 1991), such that



Grid
Draft tokens (γ)
Batch size (B)
$(0,0)$ $(1,0)$ $\dots$ $(B,0)$
$(0,1)$ $(1,1)$ $\dots$ $\vdots$
$(0,\gamma)$ $(1,\gamma)$ $\dots$ (B,γ)
Block
Threads ($\mathcal{V}_k$)
Vocabulary ($\mathcal{V}$)
1
Load partial data into SRAM.
HBM
$p(x|x_{<i+c})$
$p_k(x)$
$q(x|x_{<i+c})$
$q_k(x)$
2
Compute in SRAM.
$f_k(x) = p_k(x) - q_k(x)$
$\tau_{c_k}(x) = \min\left(1, \frac{p_k(x)}{q_k(x)}\right)$
$a_k(x) = \max(0, f_k(x))$
$b_k = \sum_{x' \in \mathcal{V}_k} \max(0, f_k(x'))$
3
Copy partial results to HBM.
HBM
$\tau_{c_k}(x)$
$\tau_c(x)$
$a_k(x)$
$a(x)$
b_k
$b = b + b_k$

Figure 1: Overview of our exact optimization approach. We compute most of the results required for speculative sampling in parallel using fast SRAM to read and write intermediate results. We maximize the number of threads per block to run parallel computation on as many elements as possible without exhausting the available on-chip memory.

each thread block computes a tile of n elements from each matrix at once. Threads within a block jointly load segments of the probability matrices $p(x|x_{x<i+c})$ and $q(x|x_{<i+c})$, whose full dimensions are $B \times \gamma \times |\mathcal{V}|$, into SRAM, effectively distributing the overhead of load latency (Ryoo et al., 2008). B denotes the batch size used during speculative decoding.

Fig. 1 illustrates the details of our approach. The overall workload is distributed across a two-dimensional grid of batch size B and number of draft tokens γ. The vocabulary $\mathcal{V}$ is partitioned into disjoint subsets $\{\mathcal{V}_k\}_{k=1}^{K}$, where $K = \lceil|\mathcal{V}|/n\rceil$ and $\lceil\cdot\rceil$ is the ceiling function. Each thread in a block performs operations on the sub-vocabulary $\mathcal{V}_k$ within its corresponding tile. The results in each tile k are obtained in three steps. 1 The domains of the probability density functions $p(x|x_{i+c})$ and $q(x|x_{i+c})$ are restricted to the sub-vocabulary $\mathcal{V}_k$ and the restricted functions are denoted as $p_k(x)$ and $q_k(x)$, respectively. All function values of $p_k(x)$ and $q_k(x)$ are loaded from HBM to SRAM.

2 All necessary partial results are computed with respect to the sub-vocabulary $\mathcal{V}_k$ and stored in SRAM. Note that the probability ratio $\tau_c(x)$ in Eq. 1, the difference $f(x) = p(x|x_{<i+c}) - q(x|x_{<i+c})$ in Eq. 2, and the numerator $a(x) = \max(0, f(x))$ in Eq. 3 can be computed in element-wise fashion. Their partial evaluations with the sub-vocabulary $\mathcal{V}_k$, denoted as $\tau_{c_k}(x)$, $f_k(x)$, and $a_k(x)$, respectively, do not require any synchronization between threads and blocks, thus allowing fast parallel computation. The denominator in Eq. 3, $b = \sum_{x' \in \mathcal{V}} \max(0, f(x')$, requires a reduction across all elements in the vocabulary $\mathcal{V}$ and is more challenging to fully compute in parallel, due to its dependency on other thread blocks. We use parallel reduction (Harris, 2007) to compute the partial sum $b_k = \sum_{x' \in \mathcal{V}_k} \max(0, f(x'))$ of the denominator with the sub-vocabulary $\mathcal{V}_k$ in SRAM, and perform the final aggregation across blocks in the subsequent procedure on HBM.

3 The partial results, τ_{c_k}, $a_k(x)$, and b_k, are written back to HBM. The partial sum b_k is now combined with the partial sums from other thread blocks to compute the full sum b. The final division operation to compute `max_norm`$(f(x))$ in Eq. 3 and the resampling procedure in Eq. 2 are done once all the partial results are aggregated.

By reorganizing the computations as illustrated in Fig. 1, batches of $p(x|x_{<i+c})$ and $q(x|x_{<i+c})$ are loaded only once from HBM into SRAM. Moreover, most operations are coupled within the kernel and performed in parallel while using fast SRAM to store intermediate values. Only the results necessary to produce token acceptance decisions are written to HBM.

3.2.2 Approximated optimization

To further accelerate speculative sampling, we use sigmoid to approximate $p(x|x_{<i+c})$ and $q(x|x_{i+c})$, which are parameterized by softmax in the baseline implementation and the exact method described in Section 3.2.1. Instead of treating $p(x|x_{<i+c})$ and $q(x|x_{i+c})$ as precomputed inputs to the kernel, the sigmoid approximation is tightly coupled with the other operations in the speculative sampling process. This integration within the kernel substantially accelerates the overall sampling procedure.

Bottleneck of softmax. For any given input vector $\mathbf{w} = (w_1, \dots, w_{|\mathcal{V}|})$, the outputs must be positive and they must sum to unity to be interpretable as a probability distribution (Bridle, 1989). In softmax, both conditions are satisfied via a normalized exponential transformation. With limited value ranges that can be represented in hardware, softmax is prone to overflow or underflow due to the exponentiation. Therefore, a numerically stable version is often used (Milakov and Gimelshein,

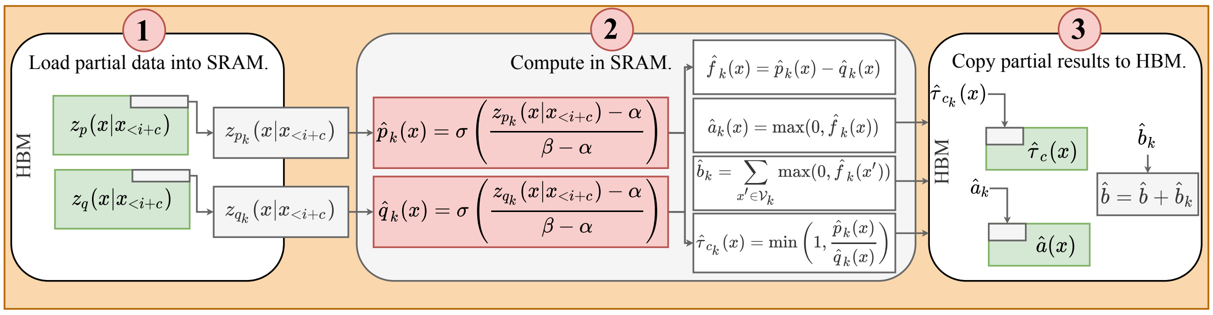

1
Load partial data into SRAM.
HBM
$z_p(x|x_{<i+c})$
$z_q(x|x_{<i+c})$
$z_{p_k}(x|x_{<i+c})$
$z_{q_k}(x|x_{<i+c})$
2
Compute in SRAM.
$\hat{p}_k(x) = \sigma\left(\frac{z_{p_k}(x|x_{<i+c}) - \alpha}{\beta - \alpha}\right)$
$\hat{q}_k(x) = \sigma\left(\frac{z_{q_k}(x|x_{<i+c}) - \alpha}{\beta - \alpha}\right)$
$\hat{f}_k(x) = \hat{p}_k(x) - \hat{q}_k(x)$
$\hat{a}_k(x) = \max(0, \hat{f}_k(x))$
$\hat{b}_k = \sum_{x' \in \mathcal{V}_k} \max(0, \hat{f}_k(x'))$
$\hat{\tau}_{c_k}(x) = \min\left(1, \frac{\hat{p}_k(x)}{\hat{q}_k(x)}\right)$
3
Copy partial results to HBM.
HBM
$\hat{\tau}_{c_k}(x)$
$\hat{\tau}_c(x)$
$\hat{a}_k$
$\hat{a}(x)$
$\hat{b}_k$
$\hat{b} = \hat{b} + \hat{b}_k$

Figure 2: Overview of the computations within each thread block for sigmoid approximation. Each set of logits is scaled by a minimum constant α and a maximum constant β. Sigmoid activations σ are then computed and stored in SRAM for each segment of draft and target logits. Subsequently, the intermediate values $\hat{f}_k(x)$, $\hat{a}_k(x)$, $\hat{b}_k$, and $\hat{\tau}_{c_k}(x)$ are computed analogous to Fig. 1. The resulting outputs are then used to update $\hat{\tau}_c(x)$, $\hat{a}(x)$, and $\hat{b}$ in HBM.

2018):

$$\texttt{softmax}(\mathbf{w})_j = \frac{\exp\left(w_j - w_{\texttt{max}}\right)}{\sum_{l=1}^{|\mathcal{V}|} \exp(w_l - w_{\texttt{max}})} \qquad (4)$$

for $j = 1, \ldots, |\mathcal{V}|$, where $w_{\texttt{max}} = \max\{w_l : 1 \leq l \leq |\mathcal{V}|\}$. Eq. 4 requires summing over the size of the vocabulary and finding $w_{\texttt{max}}$, which makes parallelization on GPUs challenging, since both the summation and $w_{\texttt{max}}$ require keeping track of intermediate values across blocks (Dao et al., 2022; Rabe and Staats, 2021; Wortsman et al., 2023).

The attention mechanism including its softmax computation has been optimized in FlashAttention (Dao et al., 2022) by fusing its operations and using an online algorithm (Milakov and Gimelshein, 2018; Rabe and Staats, 2021) that splits the workload into blocks and rescales the output of each block. Unlike FlashAttention, we explore a fully local operation that can run without expensive tracking of intermediate variables across blocks, thus allowing for non-blocking parallel computation.

Sigmoid approximation. Let $z_p(x|x_{<i+c})$ be the logits of the target model $M_p^{\texttt{target}}$ given the prefix $(x_1, \ldots, x_{i+c-1})$. Similarly, let $z_q(x|x_{<i+c})$ be the logits of the draft model $M_q^{\texttt{draft}}$. First, we rescale the logits using predefined constant values $\alpha < 0$ and $\beta > 0$, and then apply sigmoid to these scaled logits to approximate $p(x|x_{i+c})$ and $q(x|x_{<i+c})$ as follows:

$$\begin{aligned} \hat{p}(x|x_{<i+c}) &= \sigma\left(\frac{z_p(x|x_{<i+c}) - \alpha}{\beta - \alpha}\right) \\ \hat{q}(x|x_{<i+c}) &= \sigma\left(\frac{z_q(x|x_{<i+c}) - \alpha}{\beta - \alpha}\right), \end{aligned} \qquad (5)$$

where $\sigma(x) = 1/(1 + \exp(-x))$. Although all values of $\hat{p}(x|x_{<i+c})$ and $\hat{q}(x|x_{<i+c})$ are positive, they do not sum to 1 and thus do not constitute valid probability distributions. Nonetheless, we rely on these approximations with the hope that they are sufficiently accurate for guiding token acceptance decisions. Using the approximations $\hat{p}$ and $\hat{q}$, we accept the draft token x_{i+c} sampled from $M_q^{\texttt{draft}}$ if $r_c \leq \hat{\tau}_c(x_{i+c})$:

$$\hat{\tau}_c(x_{i+c}) = \min\left(1, \frac{\hat{p}(x_{i+c}|x_{<i+c})}{\hat{q}(x_{i+c}|x_{<i+c})}\right)$$
$$r_c \sim U(0, 1),$$

which is the approximation of Eq. 1. If the token x_{i+c} is rejected, we resample a token from a distribution that approximates Eq. 2:

$$x_{i+c} \sim \texttt{max\_norm}\left(\hat{p}(x|x_{<i+c}) - \hat{q}(x|x_{<i+c})\right).$$

Fig. 2 illustrates the computations with sigmoid approximation executed in parallel within each thread block. The main changes are highlighted in red rectangles. ① Let $z_{p_k}(x|x_{<i+c})$ and $z_{q_k}(x|x_{<i+c})$ be restrictions of the logits $z_p(x|x_{<i+c})$ and $z_q(x|x_{<i+c})$ to the sub-vocabulary $\mathcal{V}_k$ of the corresponding current tile. The function values of $z_{p_k}(x|x_{<i+c})$ and $z_{q_k}(x|x_{<i+c})$ evaluated on $\mathcal{V}_k$ are loaded from HBM into SRAM. ② We apply sigmoid to the rescaled logits $z_{p_k}(x|x_{<i+c})$ and $z_{q_k}(x|x_{<i+c})$. Since the computation of sigmoid is an element-wise operation and does not depend on values from other threads and blocks, we can execute it in parallel, thereby further accelerating speculative sampling. Similar to step ② of the exact optimization in Fig. 1, the partial results, $\hat{f}_k(x)$, $\hat{a}_k(x)$, $\hat{b}_k$, and $\hat{\tau}_{c_k}(x)$, which are approximations of $f_k(x)$, $a_k(x)$, b_k, and $\tau_{c_k}(x)$, respectively, are computed and stored in SRAM. ③ The partial results are written back to HBM, and for $\hat{b}_k$, they are aggregated across blocks to compute the final result $\hat{b}$, which is approximation of b.

4 Experiments

4.1 Experimental setup

Datasets and metrics. We evaluate accuracy and inference speed of our optimized speculative sampling on ASR and single-document summarization. For ASR, we measure word error rates (WERs) on the test portions of three English datasets: CommonVoice 16 (CV16) (Ardila et al., 2020), LibriSpeech (Panayotov et al., 2015), and TED-LIUM (Rousseau et al., 2012). For summarization, we use the test portions of Extreme Summarization (Xsum) (Narayan et al., 2018) and CNN/Daily Mail (CNN/DM) (Nallapati et al., 2016) to evaluate the quality of summaries generated by language models with ROUGE-1 (Lin, 2004). Additional dataset details are provided in Appendix A.3.

For all tasks, we use the PyTorch (Paszke et al., 2019) profiling tool to obtain execution times for performing speculative sampling. We measure the execution time within the entire call stack of the speculative sampling function, including any nested function call (*e.g.* softmax). The profiling times are summed over all decoding steps and examples in a dataset, before the relative improvement is calculated.

Hyperparameters. We set the batch size B to 1 and employ the same heuristic used in the baseline speculative sampling implementation in the Transformers library (Wolf et al., 2020), to determine the number of draft tokens γ. Initially, γ is set to 5 and increases by 2 if all speculative tokens sampled from the draft model are accepted; otherwise, it decreases by 1. We set the maximum sequence length to 256 tokens for ASR and 100 tokens for summarization. For ASR, using sigmoid approximation, α and β are set to -10^3 and 10^3, respectively. In summarization experiments, we use $\alpha = -10^4$ and $\beta = 10^4$. We set $n = 1024$, *i.e.*, the maximum available threads per block on the NVIDIA A100 GPU.

Target models. We employ Whisper (Radford et al., 2022) as the target model series for the ASR task. We use both the multilingual 1.55B parameter `whisper-large-v2` version and the English-only 244M parameter `whisper-small.en` version of the model. For the summarization task, we use Llama2 7B/13B (Touvron et al., 2023b), Qwen 1.8B/7B (Bai et al., 2023a), and Gemma 7B (Mesnard et al., 2024). More details on the target models are provided in Appendix A.1.

Draft models. Following Leviathan et al. (2023), Chen et al. (2023a), and Zhou et al. (2024) we either use smaller models of the same series or distilled versions of the target model for drafting. The draft model family for the ASR task is Distil-Whisper (Gandhi et al., 2023). In particular, we use the 166M parameter `small.en` and the 756M parameter `distil-large-v2` versions. The draft model for Llama2 is Sheared-LLaMA (Xia et al., 2024b), a version of Llama2 pruned to 1.3B parameters. The draft models corresponding to Qwen and Gemma are the 500M and 2B parameter versions of the same series. More details on the draft models are provided in Appendix A.2.

Implementation details. We use the implementation of speculative sampling provided by the Transformers library (Wolf et al., 2020) (v4.38.2) in conjunction with PyTorch (v2.2.2) as our baseline. Unless stated otherwise, all models are loaded in FP16 and executed on A100 GPUs with 80GB HBM using the same compute node running CUDA 12.3 and NVIDIA device driver version 545.

4.2 Main results

Table 1 summarizes accuracy metrics and profiling results for the ASR and text summarization tasks. The table details the datasets, target and draft models used, performance metrics, and the relative reduction in overall GPU profiling time achieved by our optimized approaches (exact and sigmoid approximation) compared to the baseline.

In the ASR task, our exact optimization method maintains the same WER compared to the baseline and achieves reduction in profiling time ranging from 8.7% to 12.5%. The sigmoid approximation approach results in moderately increased WERs, but yields more significant profiling time improvements, ranging from 71.9% to 84.0%.

In the text summarization task, our exact optimization method demonstrates a similar trend as observed in the previous ASR experiments, reducing profiling time by 5.7% to 11.1% without affecting ROUGE-1 scores. The non-exact sigmoid approximation further achieves significant profiling time reductions, reaching up to 93.6%. However, we also observe an absolute difference in ROUGE-1 of 0.02 to 0.06 points.

Additionally, we provide relative wall-clock time improvements for the overall text generation process in Table 5 of Appendix A.5, showing that the results obtained via profiling translate into improvements in wall-clock time for both the exact and the

Dataset	Subset	Model		WER (↓)			Δ% Profiling Time	
		Target	Draft	Baseline	Exact	Sigmoid	Exact	Sigmoid
LibriSpeech	clean	Whisper Small.EN	Distil-Whisper Small.EN	0.08	0.08	0.09	11.7%	71.9%
	other			0.14	0.14	0.15	10.7%	74.6%
TED-LIUM	release3			0.22	0.22	0.24	12.5%	78.9%
CV16	en			0.22	0.22	0.27	10.4%	66.8%
LibriSpeech	clean	Whisper Large V2	Distil-Whisper Large V2	0.07	0.07	0.15	10.8%	83.9%
	other			0.12	0.12	0.19	11.4%	84.0%
TED-LIUM	release3			0.20	0.20	0.23	11.4%	83.3%
CV16	en			0.25	0.25	0.31	8.7%	78.5%

Dataset	Subset	Model		ROUGE-1 (↑)			Δ% Profiling Time	
		Target	Draft	Baseline	Exact	Sigmoid	Exact	Sigmoid
CNN/DM	–	Llama2 7B	Sheared Llama 1.3B	0.30	0.30	0.26	5.7%	78.0%
	–	Llama2 13B	Sheared Llama 1.3B	0.31	0.31	0.29	10.1%	37.2%
	–	Qwen 7B	Qwen 0.5B	0.31	0.31	0.26	6.8%	92.4%
	–	Gemma 7B	Gemma 2B	0.23	0.23	0.17	10.6%	68.4%
Xsum	–	Llama2 7B	Sheared Llama 1.3B	0.20	0.20	0.18	11.1%	83.6%
	–	Llama2 13B	Sheared Llama 1.3B	0.20	0.20	0.17	10.5%	45.7%
	–	Qwen 7B	Qwen 0.5B	0.18	0.18	0.13	7.8%	93.6%
	–	Gemma 7B	Gemma 2B	0.18	0.18	0.13	9.9%	63.5%

Table 1: Accuracy and profiling results on ASR and text summarization. The column "Δ% Profiling Time" measures the relative reduction in GPU time achieved with our optimized approaches (*exact* and *sigmoid approximation*) compared to the baseline.

sigmoid approximation method.

4.3 Analysis and discussion

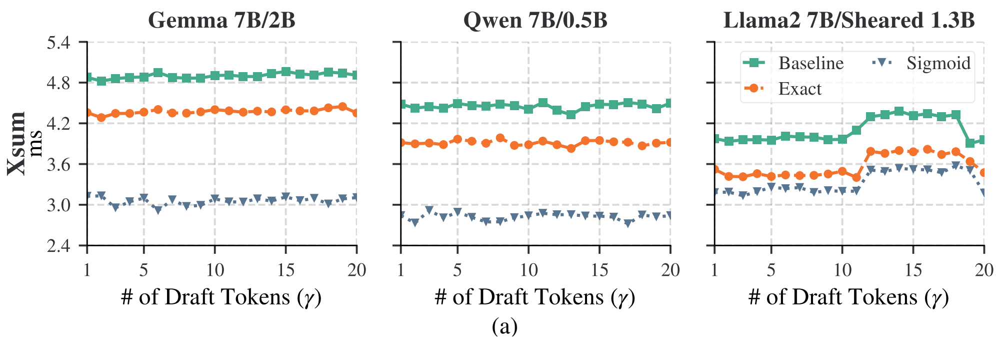
Execution times remain stable over varying γ. To assess the robustness of our exact and sigmoid optimization methods, we measure execution times across different models and varying numbers of initial draft tokens γ. For text summarization, we randomly sample 10% of the Xsum test set and use Gemma, Qwen, and Llama2 model combinations to generate summaries. For ASR, we use 10% of randomly sampled examples from the CV16 test set. Fig. 3a and Fig. 3b illustrate the average execution times of the different implementations profiled per decoding step. Both figures show average execution times measured in milliseconds (ms) for the number of draft tokens ranging from 1 to 20. The average execution times for the optimized approaches (exact and sigmoid) are consistently below the baseline across all models and values of γ. Furthermore, the execution times of the optimized approaches are stable across different choices of γ for the Gemma and Qwen models, whereas the Llama2 7B/Sheared LLaMA 1.3B model combination exhibits small sensitivity to the number of draft tokens.

As depicted in Fig. 3b, the ASR models also exhibit stable execution times across different γ, further validating the robustness of our optimization methods with varying numbers of draft tokens.

Optimized sampling does not introduce additional memory overhead. We assess the memory usage of our optimized methods relative to the baseline implementation. Fig. 4 shows the peak memory usage (HBM) on randomly sampled instances (10%) of the Xsum test set with various initial values of γ and different language models. The graph indicates that our optimized approaches do not introduce additional memory overhead compared to the baseline. For all three model combinations (Llama2, Qwen, and Gemma), the memory usage of the optimized methods fluctuates slightly (within a range of approximately 200MB) around the memory usage of the baseline across all draft tokens. As illustrated in Fig. 5, the memory usage results for the Whisper models display a pattern consistent with the text summarization experiments, showing fluctuation within a range of under 10 MB.

Effect of scaling logits. To study the effect of logit scaling in the sigmoid approximation method, we compare profiling time and performance metrics under varying values of α and β (cf. Eq. 5). Table 2 provides a comparison of different scaling factors for subsets (10%) of CV16 and Xsum using Whisper Small.EN and Llama2 7B, respectively. For each task, the table shows the values of α and β applied, the resulting WER or ROUGE-1 score, and the relative improvement in profiling time over the non-optimized baseline implementation. Note that scaling is necessary due to the numerical instability induced by the exponentiation in the sigmoid function.

The Llama2 model combination exhibits relative stability across different scaling factors, showing



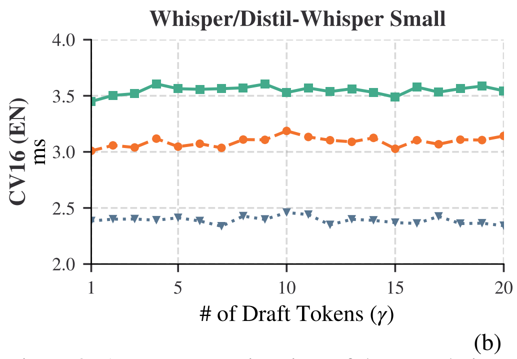


Figure 3: Average execution time of the speculative sampling algorithm per decoding step for varying initial γ values on randomly sampled subsets (10%) of Xsum and CV16 test sets.

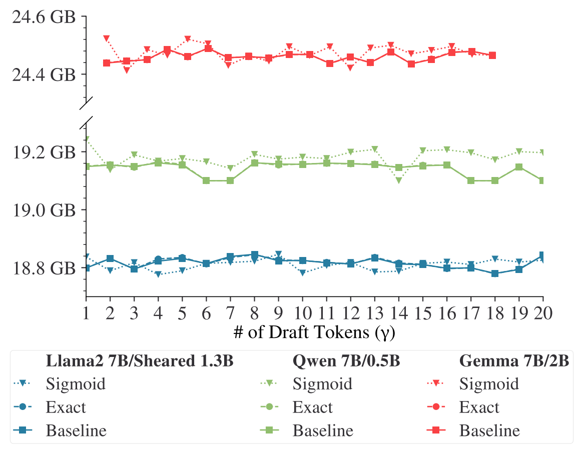


Figure 4: Peak memory usage (HBM) on randomly sampled 10% of the **Xsum** test set for varying initial values of γ.

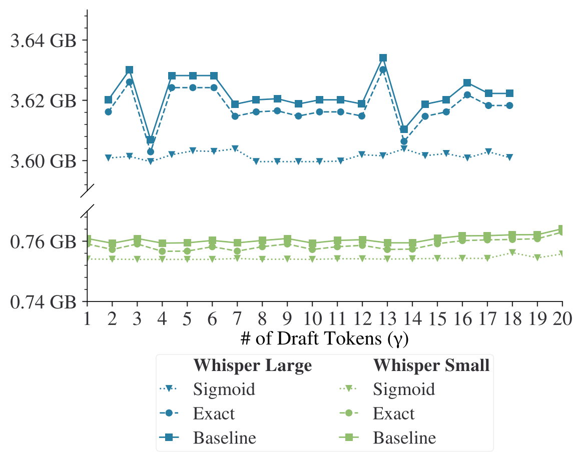


Figure 5: Peak memory usage (HBM) on randomly sampled 10% of the **CV16** test set for varying initial values of γ.

minor fluctuations in ROUGE-1 scores and profiling time improvements, whereas the Whisper model combination is more sensitive, with scaling factors of $\pm 10^5$ leading to substantial deterioration of both WER and profiling time. This can be attributed to the logits of Whisper models being generated in half precision, whereas the logits generated by the Llama models are available in full precision. However, we also find that scaling factors of $\pm 10^3$ and $\pm 10^4$ generally yield comparable results in both accuracy and profiling time improvement across the model combinations investigated in this work. The results of the same analysis for the other draft and target model combinations are provided in Table 7 from Appendix A.6.

Furthermore, we see a general trend where higher accuracy (lower WER and higher ROUGE-1) coincides with higher profiling time improvements. This relationship is due to the token acceptance process, where better calibrated models accept more tokens, requiring fewer executions of resampling and fewer overall calls of the speculative sampling kernel.

Data transfer between HBM and SRAM. To assess efficiency in terms of data movement between HBM and on-chip SRAM, we compare the realized bandwidths of each implementation. The bandwidth is calculated by dividing the total bytes transferred by the execution time. The number of

Target/Draft	Scale (α, β)	WER (↓)	Δ% Prof. Time
Whisper Small.EN/ Distil-Whisper Small.EN	Baseline	0.24	–
	-10^1 10^1	0.41	24.0%
	-10^3 10^3	0.28	59.5%
	-10^4 10^4	0.26	64.6%
	-10^5 10^5	29.34	-10826.1%

Target/Draft	Scale (α, β)	ROUGE-1 (↑)	Δ% Prof. Time
Llama2 7B/ Sheared Llama 1.3B	Baseline	0.19	–
	-10^1 10^1	0.16	49.8%
	-10^3 10^3	0.16	53.4%
	-10^4 10^4	0.15	50.0%
	-10^5 10^5	0.16	51.9%

Table 2: Impact of varying α and β on accuracy and profiling time of sigmoid approximation on **CV16** and **Xsum**.

bytes transferred is derived from the total number of sectors moved between HBM and SRAM, with each sector consisting of 32 bytes. Execution time refers to the duration during which the GPU is actively running a kernel, i.e., when at least one GPU unit is engaged in computation rather than idling, waiting, or stalled. A lower realized bandwidth indicates a reduced communication overhead between HBM and SRAM.

The results in Table 3 show lower memory bandwidth usage for the Qwen and Gemma models with the exact optimization approach compared to the baseline implementation. However, in the case of the Llama2 and Whisper models, despite overall lower realized bandwidths relative to the Qwen and Gemma models, higher bandwidths are observed with the exact optimization compared to their corresponding baseline.

The sigmoid approximation has consistently higher realized bandwidths across all model combinations compared to the baseline. Although the sigmoid optimization approach reduces the overall amount of data transferred, *i.e.*, the total number of bytes moved between HBM and SRAM, due to the element-wise approximation of the softmax function within the sampling kernel, the significantly faster execution times result in higher overall realized bandwidths. However, even the highest realized bandwidths are far below the theoretical HBM bandwidth limit of ∼2 TB/s (NVIDIA Corporation, 2020), indicating that memory transfer is not the limiting factor for performance.

Results on RTX 2080 TI GPU. Table 4 shows an accuracy and profiling time comparison using RTX 2080 TI GPUs. We observe that our optimization methods achieve performance similar to A100 GPUs with relative improvements in profiling time ranging between ∼5% and ∼13% with the exact method and between ∼62% and ∼82%

Model		Realized Bandwidth		
Draft	Target	Baseline	Exact	Sigmoid
Whisper Small.EN	Distil-Whisper Small.EN	13.98 GB/s	14.16 GB/s	16.33 GB/s
Whisper Large V2	Distil-Whisper Large V2	9.32 GB/s	11.18 GB/s	16.06 GB/s
Qwen 7B	Qwen 0.5B	44.99 GB/s	31.65 GB/s	52.14 GB/s
Gemma 7B	Gemma 2B	53.69 GB/s	38.51 GB/s	62.99 GB/s
Llama2 7B	Sheared Llama 1.3B	24.56 GB/s	27.70 GB/s	30.52 GB/s
Llama2 13B	Sheared Llama 1.3B	20.18 GB/s	24.08 GB/s	31.39 GB/s

Table 3: Comparison of realized bandwidths across models and optimization techniques using 100 examples of the XSum test set for Qwen, Gemma, and Llama2 models and 100 examples of the CV16 test set for Whisper models.

with sigmoid approximation. The summarization experiment with Qwen uses a 1.8B/0.5B parameter model combination instead of the 7B/0.5B model combination from the main experiments in Table 1, due to the limited HBM size of 11GB available on the RTX 2080 TI series.

Dataset	Subset	Target/Draft	WER (↓)			Δ Prof. Time	
			Basel.	Exact	Sigm.	Exact	Sigmoid
LibriSpeech	clean	Whisper Small.EN/ Distil Small.EN	0.08	0.08	0.09	7.6%	66.8%
LibriSpeech	other		0.14	0.14	0.15	12.3%	65.9%
TED-LIUM	release3		0.22	0.22	0.24	12.6%	62.8%
CV16	en		0.22	0.22	0.27	12.3%	62.0%
LibriSpeech	clean	Whisper Large V2/ Distil Large V2	0.07	0.07	0.15	11.2%	82.5%
LibriSpeech	other		0.12	0.12	0.19	5.7%	79.2%
TED-LIUM	release3		0.20	0.20	0.23	9.4%	79.8%
CV16	en		0.25	0.25	0.31	7.8%	75.6%
Dataset	**Subset**	**Target/Draft**	**ROUGE-1 (↑)**			**Δ Prof. Time**	
			Base.	Exact	Sigm.	Exact	Sigmoid
Xsum	–	Qwen 1.8B Qwen 0.5	0.15	0.15	0.11	5.0%	76.8%

Table 4: Accuracy and profiling time comparison across various datasets and models using a **RTX 2080 TI** GPUs.

5 Conclusions

We introduced two optimization methods to accelerate speculative sampling for autoregressive models on hardware accelerators. By computing significant portions of intermediate matrices across multiple GPU threads within thread blocks, our exact optimization method led to improved sampling speed without compromising accuracy. Additionally, we employed an approximation technique using element-wise sigmoid instead of softmax, to enable parallel computation of probabilities. This approximation further accelerated the decoding process but resulted in a small degradation of sampling quality.

Limitations

In this work, we study the inference efficiency of speech and language models in the context of speculative decoding using GPU hardware accelerators. Our investigation includes two optimized speculative sampling algorithms, tested on these models to enhance inference speed. While GPUs are the most common general-purpose hardware accelerators, there exist purpose-built architectures such as Cerebras's Wafer Scale Engines, Google's TPUs, and GraphCore's IPUs, where the differences in system design may negate or significantly reduce the latency gains. Our experiments were conducted exclusively on A100 and RTX 2080 TI GPUs on a single compute node. Therefore, the generalizability of the results to other hardware configurations remains uncertain. The performance outcomes may be influenced by other hardware and network configurations, such as multi-GPU and multi-node setups, as well as the availability fast interconnects (*e.g.* Infiniband), and other network conditions. Additionally, our study evaluates the effectiveness of the optimized algorithm based on decoding time, and our claims may not translate to other metrics, such as energy usage or heat generation, although they play an important role in real-world production settings.

Acknowledgments

We gratefully acknowledge the scientific support and HPC resources provided by the Erlangen National High Performance Computing Center (NHR@FAU) of the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) under the NHR project b196ac14. NHR funding is provided by federal and Bavarian state authorities. NHR@FAU hardware is partially funded by the German Research Foundation (DFG) – 440719683. This work was supported by the Bavarian State Ministry of Science and the Arts under grant H.2-F1116.NÜ/61/2.

References

Josh Achiam et al. 2024. GPT-4 technical report. *Preprint*, arXiv:2303.08774.

Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Rosana Ardila, Megan Branson, Kelly Davis, Michael Kohler, Josh Meyer, Michael Henretty, Reuben Morais, Lindsay Saunders, Francis Tyers, and Gregor Weber. 2020. Common Voice: A massively-multilingual speech corpus. In *LREC*, pages 4218–4222.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023a. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Yu Bai, Fan Chen, Huan Wang, Caiming Xiong, and Song Mei. 2023b. Transformers as statisticians: Provable in-context learning with in-context algorithm selection. *Advances in Neural Information Processing Systems (Neurips)*.

Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. 2023. Quantizable transformers: Removing outliers by helping attention heads do nothing. *Advances in Neural Information Processing Systems (NeurIPS)*.

John Bridle. 1989. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. *Advances in Neural Information Processing Systems (NeurIPS)*.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023a. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*.

Ke Chen, Yue Gao, Haroon Waris, Weiqiang Liu, and Fabrizio Lombardi. 2023b. Approximate softmax functions for energy-efficient deep neural networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*.

J. Cheng, M. Grossman, and T. McKercher. 2014. *Professional CUDA C Programming*. Wiley.

Ryan A Chi, Jeremy Kim, Scott Hickmann, Siyan Li, Gordon Chi, Thanawan Atchariyachanvanit, Katherine Yu, Nathan A Chi, Gary Dai, Shashank Rammoorthy, et al. 2023. Dialogue distillery: Crafting interpolable, interpretable, and introspectable dialogue from LLMs. *Alexa Prize SocialBot Grand Challenge*, 5.

Tri Dao. 2024. FlashAttention-2: Faster attention with better parallelism and work partitioning. *International Conference on Learning Representations (ICLR)*.

Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Re. 2022. Flashattention: Fast and memory-efficient exact attention with IO-awareness. *Neural Information Processing Systems (NeurIPS)*.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. GPT3.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Jared Fernandez, Jacob Kahn, Clara Na, Yonatan Bisk, and Emma Strubell. 2023. The framework tax: Disparities between inference efficiency in NLP research and deployment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1588–1600, Singapore. Association for Computational Linguistics.

Sophie Fischer, Carlos Gemmell, Niklas Tecklenburg, Iain Mackie, Federico Rossetto, and Jeffrey Dalton. 2024. GRILLBot in practice: Lessons and tradeoffs deploying large language models for adaptable conversational task assistants. *arXiv preprint arXiv:2402.07647*.

Sanchit Gandhi, Patrick von Platen, and Alexander M Rush. 2023. Distil-whisper: Robust knowledge distillation via large-scale pseudo labelling. *arXiv preprint arXiv:2311.00430*.

Xue Geng, Jie Lin, Bin Zhao, Anmin Kong, Mohamed M. Sabry Aly, and Vijay Ramaseshan Chandrasekhar. 2018. Hardware-aware softmax approximation for deep neural networks. *Asian Conference on Computer Vision*.

Édouard Grave, Armand Joulin, Moustapha Cissé, David Grangier Facebook AI Research, and Hervé Jégou. 2017. Efficient softmax approximation for GPUs. *International Conference on Machine Learning (ICML)*.

Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A Roberts. 2024. The unreasonable ineffectiveness of the deeper layers. *arXiv preprint arXiv:2403.17887*.

Mark Harris. 2007. Optimizing parallel reduction in cuda. Technical report, NVIDIA. Accessed: 2024-06-12.

Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*.

Jiri Hron, Yasaman Bahri, Jascha Narain Sohl-Dickstein, and Roman Novak. 2020. Infinite attention: Nngp and ntk for deep attention networks. *International Conference on Machine Learning (ICML)*.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8003–8017, Toronto, Canada. Association for Computational Linguistics.

Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. 2021. Hubert: Self-supervised speech representation learning by masked prediction of hidden units. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, page 3451–3460.

Weizhe Hua, Zihang Dai, Hanxiao Liu, and Quoc Le. 2022. Transformer quality in linear time. *International Conference on Machine Learning (ICML)*.

Zhe Jia, Marco Maggioni, Benjamin Staiger, and Daniele P. Scarpazza. 2018. Dissecting the NVIDIA volta GPU architecture via microbenchmarking. *arxiv preprint arXiv:1804.06826*.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online. Association for Computational Linguistics.

François Lagunas, Ella Charlaix, Victor Sanh, and Alexander Rush. 2021. Block pruning for faster transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10619–10629, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Monica D. Lam, Edward E. Rothberg, and Michael E. Wolf. 1991. The cache performance and optimizations of blocked algorithms. *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

Changhyeon Lee and Seulki Lee. 2023. Softmax output approximation for activation memory-efficient training of attention-based networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 36.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. *International Conference on Machine Learning (ICML)*.

Zhiyuan Li, Srinadh Bhojanapalli, Manzil Zaheer, Sashank Reddi, and Sanjiv Kumar. 2022. Robust training of neural networks using scale invariant architectures. *International Conference on Machine Learning (ICML)*.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. 2024. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. 2024. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*.

Maxim Milakov and Natalia Gimelshein. 2018. Online normalizer calculation for softmax. *arXiv preprint arXiv:1805.02867*.

Giovanni Monea, Armand Joulin, and Edouard Grave. 2023. PaSS: Parallel speculative sampling. *arXiv preprint arXiv:2311.13581*.

Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart Van Baalen, and Tijmen Blankevoort. 2021. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar Gulçehre, and Bing Xiang. 2016. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, Berlin, Germany. Association for Computational Linguistics.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807, Brussels, Belgium. Association for Computational Linguistics.

Radford M. Neal. 2003. Slice sampling. *The Annals of Statistics*, 31(3).

NVIDIA Corporation. 2020. NVIDIA A100 tensor core GPU architecture. Technical report, NVIDIA Corporation.

Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. 2015. Librispeech: An asr corpus based on public domain audio books. In *ICASSP*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. *Neural Information Processing Systems (NeurIPS)*.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. *Machine Learning and Systems*.

Markus N Rabe and Charles Staats. 2021. Self-attention does not need $o(n^2)$ memory. *arXiv preprint arXiv:2112.05682*.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. 2022. Robust speech recognition via large-scale weak supervision. *Preprint*, arXiv:2212.04356. ArXiv:2212.04356.

Jason Ramapuram, Federico Danieli, Eeshan Dhekane, Floris Weers, Dan Busbridge, Pierre Ablin, Tatiana Likhomanenko, Jagrit Digani, Zijin Gu, Amitis Shidani, and Russ Webb. 2024. Theory, analysis, and best practices for sigmoid self-attention. *Preprint*, arXiv:2409.04431.

Anthony Rousseau, Paul Deléglise, and Yannick Estève. 2012. TED-LIUM: an automatic speech recognition dedicated corpus. In *LREC*, pages 125–129.

Shane Ryoo, Christopher I Rodrigues, Sam S Stone, Sara S Baghsorkhi, Sain-Zee Ueng, John A Stratton, and Wen-mei W Hwu. 2008. Program optimization space pruning for a multithreaded gpu. In *IEEE/ACM international symposium on Code generation and optimization*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Noam Shazeer. 2019. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*.

Noam Shazeer. 2020. GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*.

Kai Shen, Junliang Guo, Xu Tan, Siliang Tang, Rui Wang, and Jiang Bian. 2023. A study on relu and softmax in transformer. *arXiv preprint arXiv:2302.06461*.

Kyuhong Shim, Minjae Lee, Iksoo Choi, Yoonho Boo, and Wonyong Sung. 2017. Svd-softmax: Fast softmax approximation on large vocabulary neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*.

Benjamin Spector and Chris Re. 2023. Accelerating LLM inference with staged speculative decoding. *arXiv preprint arXiv:2308.04623*.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. *Advances in Neural Information Processing Systems (NeurIPS)*.

Pierre Stock, Angela Fan, Benjamin Graham, Edouard Grave, Rémi Gribonval, Herve Jegou, and Armand Joulin. 2021. Training with quantization noise for extreme model compression. In *International Conference on Learning Representations (ICLR)*.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for BERT model compression. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4323–4332, Hong Kong, China. Association for Computational Linguistics.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Hugo Touvron et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems (NeurIPS)*.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, Florence, Italy. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Mitchell Wortsman, Jaehoon Lee, Justin Gilmer, and Simon Kornblith. 2023. Replacing softmax with relu in vision transformers. *arXiv preprint arXiv:2309.08586*.

Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. 2023. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3909–3925, Singapore. Association for Computational Linguistics.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. 2024a. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. *arXiv preprint arXiv:2401.07851*.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2024b. Sheared LLaMA: Accelerating language model pre-training via structured pruning. *International Conference on Learning Representations (ICLR)*.

Sen Yang, Shujian Huang, Xinyu Dai, and Jiajun Chen. 2024. Multi-candidate speculative decoding. *arXiv preprint arXiv:2401.06706*.

Aonan Zhang, Chong Wang, Yi Wang, Xuanyu Zhang, and Yunfei Cheng. 2024a. Recurrent drafter for fast speculative decoding in large language models. *Preprint*, arXiv:2403.09919.

Biao Zhang and Rico Sennrich. 2019. Root mean square layer normalization. *Neural Information Processing Systems (NeurIPS)*.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. 2024b. Draft& verify: Lossless large language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11263–11282, Bangkok, Thailand. Association for Computational Linguistics.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. 2024. Distillspec: Improving speculative decoding via knowledge distillation. In *International Conference on Learning Representations (ICLR)*.

A Appendix

A.1 Target model details

Whisper. Whisper (Radford et al., 2022) is a family of models trained to perform multiple tasks such as multilingual ASR, language identification, and speech translation. The models are trained on ∼680k hours of labeled audio data retrieved from the world wide web and are available in five sizes ranging from 39M parameters to 1.55B parameters. Utilizing an encoder-decoder Transformer architecture, Whisper receives 80-channel log-Mel spectrogram representations with a 25ms window and a stride of 10ms as inputs. We conduct experiments on both the multilingual 1.55B parameter `whisper-large-v2` version and the English-only 244M parameter `whisper-small.en` version.

Llama2. Llama2 (Touvron et al., 2023b) is a collection of LLMs ranging from 7B to 70B parameters. The models are pretrained on 2 trillion tokens of text data from publicly available sources. The architecture is based on Llama1 (Touvron et al., 2023a), utilizing pre-normalization with RMSNorm (Zhang and Sennrich, 2019), SwiGLU (Shazeer, 2020) activation functions, and rotary positional embeddings (RoPE) (Su et al., 2024). Notable architectural changes include an expanded context length of 4K tokens and the adoption of grouped-query attention (GQA) for the 34B and 70B models. We employ the 7B and 13B versions of Llama2 as target models.

Qwen. The Qwen (Bai et al., 2023a) model series offers a range of decoder-only language models with parameter counts between 500M and 110B. The models are pretrained on up to 3 trillion tokens of various multilingual text, code, and mathematics resources. The architecture is similar to Llama2 with small modifications, such as no weight tying between input embeddings and output projection. We employ Qwen v1.5 in our experiments and use the 7B parameter variant as the target model.

Gemma. Gemma (Mesnard et al., 2024) comprises two model variants, featuring 2B and 7B parameters, pretrained on 3 trillion and 6 trillion tokens respectively. Gemma is based on the Gemini (Anil et al., 2023) model family. The focus is primarily on English text from web documents, mathematics, and code, omitting multimodal capabilities and optimization for multilingual tasks. Similar to Llama2, the Gemma models leverage RoPE and RMSNorm, and embeddings are shared across inputs and outputs to reduce model size. We use the 7B parameter variant of Gemma v1.0 as the target model.

A.2 Draft model details

Distil-Whisper. The draft model series for the ASR task is Distil-Whisper (Gandhi et al., 2023), a collection of smaller versions of the Whisper model. Distil-Whisper applies knowledge distillation (Hinton et al., 2015) to emulate the performance of the original Whisper model using a large (≈21k hours) pseudo-labeled training corpus. The distilled models aim to maintain the robustness of Whisper towards varying audio domains and noisy acoustic conditions and are designed to be paired with Whisper in a speculative decoding setting. We use the 166M parameter `small.en` version as the draft model for the small 244M parameter target model and the 756M parameter `distil-large-v2` version as the draft model for the large 1.55B parameter target model.

Sheared-LLaMA. The draft model series for our experiments with Llama2 is Sheared-LLaMA (Xia et al., 2024b). Sheared-LLaMA utilizes a structured pruning approach to reduce the size of the 7B parameter Llama 2 model to 1.3B and 2.7B parameters. The structured pruning approach removes parameters from the source model until a given target configuration is satisfied. Learned pruning masks representing discrete prune or retain decisions are used to create a smaller sub-network matching the specified target configuration. We employ the 1.3B version of Sheared-LLaMA in our experiments.

A.3 Dataset details

ASR. We used the test sets of three English ASR benchmark datasets: CommonVoice v16 (Ardila et al., 2020), LibriSpeech (Panayotov et al., 2015), and TED-LIUM (Rousseau et al., 2012) for the ASR task. The data comprises multiple domains such as audiobooks, political speeches, interviews, and narrated Wikipedia articles. The utterance lengths vary between 0.2 and 330 seconds with an average duration of 7.6±6.6 seconds.

Text summarization. We used two datasets for text summarization: Extreme Summarization (Xsum) (Narayan et al., 2018) and CNN/Daily Mail (CNN/DM) (Nallapati et al., 2016). The Xsum test set contains 11,334 online articles from the British Broadcasting Corporation (BBC) and the CNN/DM test set contains 11,490 news articles published by CNN and the Daily Mail. We performed 0-shot

Dataset	Subset	Model		Wall-clock Improvement	
		Target	Draft	Exact	Sigmoid
TED-LIUM	release3	Whisper Small.EN	Distil-Whisper Small.EN	8.7%	58.7%
	release3	Whisper Large V2	Distil-Whisper Large V2	6.6%	66.3%
LibriSpeech	clean	Whisper Small.EN	Distil-Whisper Small.EN	5.8%	49.5%
	clean	Whisper Large V2	Distil-Whisper Large V2	7.4%	65.3%
	other	Whisper Small.EN	Distil-Whisper Small.EN	7.1%	54.5%
	other	Whisper Large V2	Distil-Whisper Large V2	7.0%	66.2%
CV16	en	Whisper Small.EN	Distil-Whisper Small.EN	8.1%	51.8%
	en	Whisper Large V2	Distil-Whisper Large V2	5.4%	61.3%
CNN/DailyMail	–	Gemma 7B	Gemma 2B	3.0%	24.2%
	–	Qwen 7B	Qwen 0.5B	1.6%	39.8%
	–	Llama2 7B	Sheared Llama 1.3B	4.2%	18.2%
	–	Llama2 13B	Sheared Llama 1.3B	2.6%	23.5%
Xsum	–	Gemma 7B	Gemma 2B	1.2%	18.5%
	–	Qwen 7B	Qwen 0.5B	4.3%	59.1%
	–	Llama2 7B	Sheared Llama 1.3B	6.5%	53.1%
	–	Llama2 13B	Sheared Llama 1.3B	10.9%	23.6%

Table 5: Relative wall-clock time improvements for both exact and sigmoid sampling on all tasks and model combinations. Wall-clock time measures the total time spent in the speculative decoding loop, including all forward passes through the draft and target models. The relative improvements are computed based on the total time required to perform speculative decoding for the full dataset.

evaluation for CNN/DM and Xsum, and used the ROUGE-1 metric for comparison. To prompt the model for a summary, we placed "`Summary:`" after each input article. Summaries were generated with a maximum token length of 100 for both Xsum and CNN/DM.

A.4 Wall-clock time improvement

Table 5 summarizes the relative wall-clock time improvements for the overall text generation process. Both the exact and the sigmoid approximation method translate into relative improvements compared to the baseline implementation. Wall-clock times are less precise, since they also include the forward passes through the draft and target models, which may lead to additional overhead introduced by the deep learning framework (Fernandez et al., 2023), and the time spent on CPU, which does not take varying rates of context switches and stalling due to execution of higher-priority processes into account.

A.5 Average times per decoding step

The average times spent in the speculative sampling procedure per decoding step are summarized in Table 6. Our implementation achieved consistently lower average sampling times than the reference implementation. While the the average sampling time was generally longer for the text generation tasks, the average times with our implementation were still consistently lower than the reference implementation.

A.6 Effect of scaling logits

Table 7 shows the impact of various logit scaling factors on performance and profiling time of sigmoid approximation on CV16 and Xsum. The values are computed on a random sample of 10% of each dataset.

A.7 Relation to other optimization methods

Our method is orthogonal to other optimizations of speculative decoding. Whenever speculative sampling is used, our kernel can serve as a drop-in replacement for the standard implementation. For example, our proposed method can be integrated with the recently proposed self-speculative decoding approach (Zhang et al., 2024b). Instead of using a separate draft model, self-speculative decoding samples draft tokens by skipping some layers of the target model. Afterwards, it follows the same draft verification and resampling procedure as the original speculative decoding, which can be further accelerated with our optimization method.

Our method is also orthogonal to other approaches for accelerating decoding. For instance, FlashAttention (Dao et al., 2022; Dao, 2024), which focuses on optimizing the attention computation, can be easily combined with our method to further improve efficiency.

A.8 Overhead caused by resampling

To assess the overhead caused by resampling, we followed Chen et al. (2023a) and computed average acceptance rates of draft tokens for various models on 10% of Xsum. Table 8 includes the acceptance rates using a varying number of draft

Dataset	Subset	Model		Avg.±Std. (ms)			Δ% Prof. Time	
		Target	Draft	Baseline	Exact	Sigmoid	Exact	Sigmoid
TED-LIUM	release3	Small.EN	Distil Small.EN	4.17±0.81	3.67±0.64	3.12±1.09	11.9%	25.1%
	release3	Large V2	Distil Large V2	4.37±0.58	3.88±0.46	3.62±1.09	11.2%	17.2%
LibriSpeech	clean	Small.EN	Distil Small.EN	4.31±1.00	3.81±0.80	3.15±1.13	11.7%	27.0%
	clean	Large V2	Distil Large V2	4.35±0.60	3.88±0.52	3.55±1.08	10.8%	18.4%
	other	Small.EN	Distil Small.EN	4.14±0.84	3.68±0.68	3.20±1.07	11.1%	22.7%
	other	Large V2	Distil Large V2	4.39±0.61	3.90±0.49	3.55±1.07	11.2%	19.1%
CV16	en	Small.EN	Distil Small.EN	4.14±0.85	3.71±0.70	3.33±1.10	10.4%	19.8%
	en	Large V2	Distil Large V2	4.37±0.61	3.92±0.51	3.62±1.07	10.4%	17.2%
CNN/DailyMail	–	Gemma 7B	Gemma 2B	6.54±0.47	5.84±0.39	4.37±0.54	10.6%	33.1%
	–	Qwen 7B	Qwen 0.5B	12.01±1.08	11.20±1.03	3.34±0.45	6.8%	72.2%
	–	Llama2 7B	Sheared Llama 1.3B	11.33±0.94	10.69±0.83	3.65±0.62	5.7%	67.8%
	–	Llama2 13B	Sheared Llama 1.3B	3.99±0.52	3.59±1.70	3.26±0.26	10.1%	18.3%
Xsum	–	Gemma 7B	Gemma 2B	6.39±0.50	5.76±0.42	4.61±0.55	9.9%	27.9%
	–	Qwen 7B	Qwen 0.5B	11.55±1.39	10.65±1.51	3.20±0.42	7.8%	72.3%
	–	Llama2 7B	Sheared Llama 1.3B	4.66±0.42	4.14±2.88	3.64±0.56	11.1%	21.8%
	–	Llama2 13B	Sheared Llama 1.3B	4.67±0.41	4.17±1.70	3.70±0.44	10.5%	20.7%

Table 6: Average time and standard deviation spent within the speculative sampling algorithm **per decoding step**. The column "Δ% Prof. Time" measures the relative reduction in average time per decoding step ("Baseline" vs. "Exact" and"Sigmoid"). Scaling constants for sigmoid approximation: $\alpha = -10^3$ and $\beta = 10^3$ for ASR, $\alpha = -10^4$ and $\beta = 10^4$ for summarization.

tokens ($\gamma \in \{3, 5, 10, 15\}$), as well as the average execution time per decoding step of the speculative sampling algorithm. Since our exact optimization method aims to generate the same tokens as the baseline implementation, we expect the acceptance rates to be the same. Table 8 shows that this is indeed the case for all choices of γ and model combinations. Table 8 also shows that the acceptance rates with the sigmoid optimization method are often higher than the acceptance rates of the baseline and the exact method. However, these higher acceptance rates, do not have a significant effect on the average execution time. In particular, the acceptance rates of the three methods (sigmoid, exact, and baseline) are similar for the Qwen model combination ($\sim$48% with $\gamma = 10$), but the sigmoid approximation achieves better execution times than the exact optimization and the baseline. The average execution times provided Table 8 are consistent with the ones presented in Fig. 3a.

Draft/Target	**Scale** (α, β)	**WER** (↓)	**Prof. Time** Δ%
Whisper Small.EN Distil Small.EN	Baseline	0.24	–
	-10^1 10^1	0.41	24.0%
	-10^3 10^3	0.28	59.5%
	-10^4 10^4	0.26	64.6%
	-10^5 10^5	29.34	-10826.1%
Whisper Large V2/ Distil Large V2	Baseline	0.24	–
	-10^1 10^1	0.42	64.3%
	-10^3 10^3	0.34	75.0%
	-10^4 10^4	0.31	78.6%
	-10^5 10^5	30.91	-4458.3%

Draft/Target	**Scale** (α, β)	**ROUGE-1** (↑)	**Prof. Time** Δ%
Gemma 7B/ Gemma 2B	Baseline	0.17	–
	-10^1 10^1	0.01	-124.6%
	-10^3 10^3	0.13	66.1%
	-10^4 10^4	0.13	71.3%
	-10^5 10^5	0.14	73.0%
Qwen 7B/ Qwen 0.5B	Baseline	0.18	–
	-10^1 10^1	0.09	57.6%
	-10^3 10^3	0.12	71.4%
	-10^4 10^4	0.12	71.5%
	-10^5 10^5	0.11	71.7%
Llama2 7B/ Sheared 1.3B	Baseline	0.19	–
	-10^1 10^1	0.16	49.8%
	-10^3 10^3	0.16	53.4%
	-10^4 10^4	0.15	50.0%
	-10^5 10^5	0.16	51.9%
Llama2 13B/ Sheared 1.3B	Baseline	0.20	–
	-10^1 10^1	0.15	46.9%
	-10^3 10^3	0.17	46.8%
	-10^4 10^4	0.15	46.7%
	-10^5 10^5	0.16	45.9%

Table 7: Impact of varying scaling factors α and β on performance and profiling time of sigmoid approximation on **CV16** and **Xsum**. The values are computed on a random sample of 10% of each dataset.

Target Model	**Draft Model**	**Method**	**Acceptance Rate**				**Average Execution Time**			
			$\gamma = 3$	$\gamma = 5$	$\gamma = 10$	$\gamma = 15$	$\gamma = 3$	$\gamma = 5$	$\gamma = 10$	$\gamma = 15$
Gemma 7B	Gemma 2B	Sigmoid	56.2%	57.0%	57.1%	56.1%	2.95 ms	3.10 ms	3.09 ms	3.12 ms
		Exact	48.5%	48.4%	48.6%	46.4%	4.35 ms	4.36 ms	4.40 ms	4.40 ms
		Baseline	48.5%	48.4%	48.6%	46.4%	4.86 ms	4.88 ms	4.90 ms	4.96 ms
Qwen 7B	Qwen 0.5B	Sigmoid	46.2%	47.8%	48.1%	47.8%	2.92 ms	2.89 ms	2.84 ms	2.83 ms
		Exact	46.8%	45.7%	48.4%	48.0%	3.91 ms	3.97 ms	3.88 ms	3.95 ms
		Baseline	46.8%	45.7%	48.4%	48.0%	4.45 ms	4.49 ms	4.41 ms	4.48 ms
Llama2 7B	Sheared Llama 1.3B	Sigmoid	58.4%	59.3%	58.1%	59.4%	3.13 ms	3.26 ms	3.20 ms	3.52 ms
		Exact	51.5%	53.7%	56.2%	54.4%	3.41 ms	3.42 ms	3.49 ms	3.78 ms
		Baseline	51.5%	53.7%	56.2%	54.4%	3.96 ms	3.95 ms	3.96 ms	4.31 ms

Table 8: Comparison of acceptance rates by optimization type for different models on a random sample of 10% of the Xsum dataset with varying γ.