MEDUSA: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads

Tianle Cai[*1,2], **Yuhong Li**[*3], **Zhengyang Geng**[4], **Hongwu Peng**[5], **Jason D. Lee**[1], **Deming Chen**[3], **Tri Dao**[1,2]

[1]Princeton University, [2]Together AI, [3]University of Illinois Urbana-Champaign, [4]Carnegie Mellon University, [5]University of Connecticut

Abstract

The inference process in Large Language Models (LLMs) is often limited due to the absence of parallelism in the auto-regressive decoding process, resulting in most operations being restricted by the memory bandwidth of accelerators. While methods such as speculative decoding have been suggested to address this issue, their implementation is impeded by the challenges associated with acquiring and maintaining a separate draft model. In this paper, we present MEDUSA, an efficient method that augments LLM inference by adding extra decoding heads to predict multiple subsequent tokens in parallel. Using a *tree-based attention mechanism*, MEDUSA constructs multiple candidate continuations and verifies them simultaneously in each decoding step. By leveraging parallel processing, MEDUSA introduces only minimal overhead in terms of single-step latency while substantially reducing the number of decoding steps required.

We present two levels of fine-tuning procedures for MEDUSA to meet the needs of different use cases:

- MEDUSA-1: MEDUSA is directly fine-tuned on top of a *frozen* backbone LLM, enabling lossless inference acceleration.
- MEDUSA-2: MEDUSA is fine-tuned together with the backbone LLM, enabling better prediction accuracy of MEDUSA heads and higher speedup but needing a special training recipe that preserves the backbone model's capabilities.

Moreover, we propose several extensions that improve or expand the utility of MEDUSA, including a *self-distillation* to handle situations where no training data is available and a *typical acceptance scheme* to boost the acceptance rate while maintaining generation quality.

We evaluate MEDUSA on models of various sizes and training procedures. Our experiments demonstrate that MEDUSA-1 can achieve over 2.2× speedup without compromising generation quality, while MEDUSA-2 further improves the speedup to 2.3-3.6×. The code for this implementation is available at `https://github.com/FasterDecoding/Medusa`.

[*]Equal contribution.

Contents

1 Introduction

The recent advancements in Large Language Models (LLMs) have demonstrated that the quality of language generation significantly improves with an increase in model size, reaching billions of parameters [Brown et al., 2020, Chowdhery et al., 2022, Zhang et al., 2022, Hoffmann et al., 2022, OpenAI, 2023, Google, 2023, Touvron et al., 2023]. However, this growth has led to an increase in *inference latency*, which poses a significant challenge in practical applications. From a system perspective, LLM inference is predominantly memory-bound [Shazeer, 2019, Kim et al., 2023], with the main latency bottleneck stemming from accelerators' memory bandwidth rather than arithmetic computations. This bottleneck is inherent to the sequential nature of auto-regressive decoding, where each forward pass requires transferring the complete model parameters from High-Bandwidth Memory (HBM) to the accelerator's cache. This process, which generates only a single token, underutilizes the arithmetic computation potential of modern accelerators, leading to inefficiency.

To address this, one approach to speed up LLM inference involves *increasing the arithmetic intensity* (the ratio of total floating-point operations (FLOPs) to total data movement) of the decoding process and *reducing the number of decoding steps*. In line with this idea, speculative decoding has been proposed [Leviathan et al., 2022, Chen et al., 2023, Xia et al., 2023, Miao et al., 2023]. This method uses a smaller draft model to generate a sequence of tokens at each step, which is then refined by the original, larger model for acceptable continuation. However, obtaining an appropriate draft model remains challenging, and things become even harder when integrating the draft model into a distributed system [Chen et al., 2023].

Instead of using a separate draft model to sequentially generate candidate outputs, in this paper, we revisit and refine the concept of using multiple decoding heads on top of the backbone model to expedite inference [Stern et al., 2018]. We find that when applied effectively, this technique can overcome the challenges of speculative decoding, allowing for seamless integration into existing LLM systems. Specifically, we introduce MEDUSA, a method that enhances LLM inference by integrating additional decoding heads capable of concurrently predicting multiple tokens. These heads are fine-tuned in a *parameter-efficient* manner and can be added to any existing model. With no requirement for a new model, MEDUSA offers easy and automatic integration into current LLM systems, including those in distributed environments, ensuring a user-friendly experience.

2 Observations:
1. single token generation leads to restricted sequence length early stopping
2. Rejection sampling is inefficient

We further enhance MEDUSA with two key insights. Firstly, the current approach of generating a single candidate continuation at each decoding step leads to a restricted acceptance length and inefficient use of computational resources. To address this, we propose generating multiple candidate continuations using the MEDUSA heads and verifying them concurrently through a simple adjustment to the attention mask. Secondly, we can use the rejection sampling scheme similar to that used in speculative decoding to generate responses with the same distribution as the original model, but it is usually unnecessary for many LLM applications. Alternatively, we also introduce a *typical acceptance* scheme that selects *reasonable* candidates from the MEDUSA head outputs. We use temperature as a threshold to manage deviation from the original model's predictions, providing an efficient alternative to the rejection sampling method. This approach effectively addresses its limitations, such as decreased speed at higher temperatures.

To equip LLMs with predictive MEDUSA heads, we propose two distinct fine-tuning procedures tailored to various scenarios. For situations with limited computational resources or when the objective is to incorporate MEDUSA into an existing model without affecting its performance, we recommend MEDUSA-1. This method requires minimal memory and can be further optimized with quantization techniques akin to those in QLoRA [Dettmers et al., 2023], without compromising the generation quality due to the fixed backbone model. However, in MEDUSA-1, the full potential of the backbone model is not utilized. We can further fine-tune it to enhance the prediction accuracy of MEDUSA heads, which can directly lead to a greater speedup. Therefore, we introduce MEDUSA-2, which is suitable for scenarios with ample computational resources or for direct Supervised Fine-Tuning (SFT) from a base model. The key to MEDUSA-2 is a training protocol that enables joint training of the MEDUSA heads and the backbone model without compromising the model's next-token prediction capability and output quality. We propose different strategies for obtaining the training datasets depending on the model's training recipe and dataset availability. When the model is fine-tuned on a public dataset, it can be directly used for MEDUSA. If the dataset is unavailable or the model underwent a Reinforcement Learning with Human Feedback (RLHF) [Ouyang et al., 2022] process, we suggest a self-distillation approach to generate a training dataset for the MEDUSA heads.

Our experiments primarily focus on scenarios with a batch size of one, which is representative of the use case where LLMs are locally hosted for personal use[2] We test MEDUSA on models of varying sizes and training settings, including Vicuna-7B, 13B (trained with a public dataset), Vicuna-33B [Chiang et al., 2023] (trained with a private dataset[3]), and Zephyr-7B (trained with both supervised fine-tuning and alignment). MEDUSA can achieve a speedup of 2.3 to 3.6 times across different prompt types without compromising on the quality of generation.

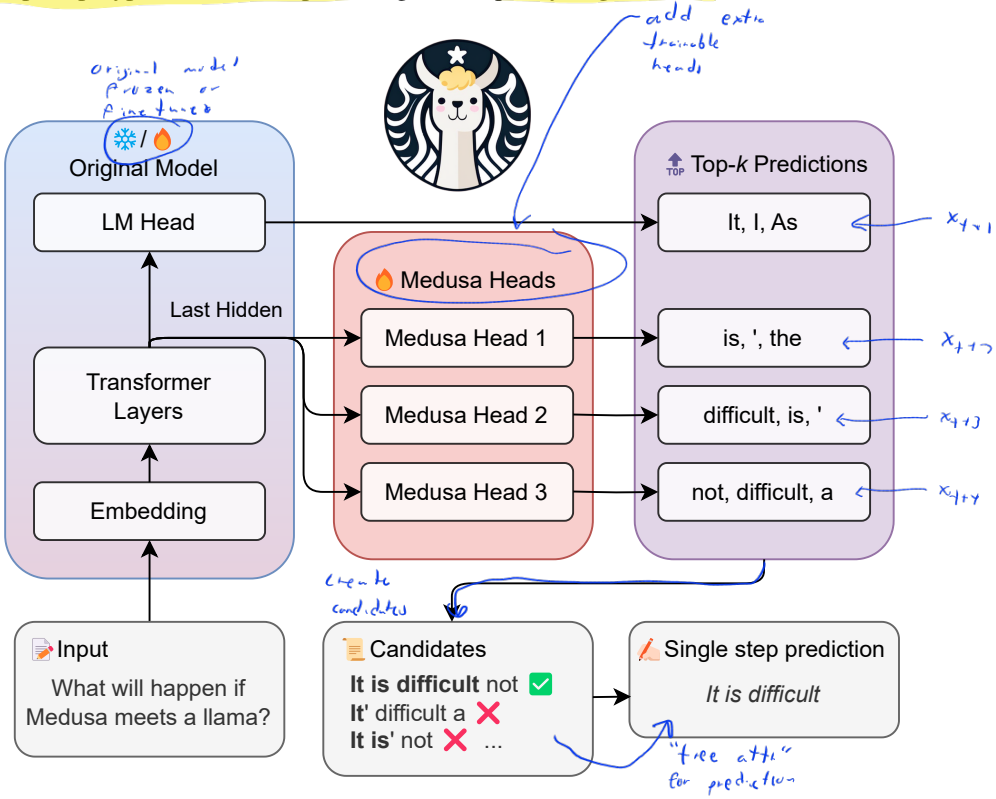


Figure 1: **Overview of MEDUSA.** MEDUSA introduces *multiple heads* on top of the last hidden states of the LLM, enabling the prediction of several subsequent tokens in parallel (Section 3.1.1). For training MEDUSA heads, the original model is either *frozen* (MEDUSA-1, Section 3.2.1) or trained together (MEDUSA-2, Section 3.2.2) with MEDUSA heads. During inference, each head generates multiple top predictions for its designated position. These predictions are assembled into candidates, which are subsequently processed in parallel using a *tree-based attention* mechanism (Section 3.1.2). The final step is to verify the candidates and accept a continuation. Besides the standard rejection sampling scheme, a *typical acceptance* scheme (Section 3.3.1) can also be used here to select reasonable continuations, and the *longest accepted candidate prefix* will be used for the next decoding phase. The efficiency of the decoding process is enhanced by accepting more tokens simultaneously, thus reducing the number of required decoding steps.

2 Related Work

2.1 LLM Inference Acceleration

The inefficiency of Large Language Model (LLM) inference is primarily attributed to the memory-bound nature of the auto-regressive decoding process. Several methods have been proposed to alleviate this issue, improving inference latency and throughput. Traditionally, batch inference has been

[2]It's important to note that while MEDUSA can be seamlessly used in a batched inference setting, it requires additional engineering efforts to integrate into a serving engine like vLLM [Kwon et al., 2023]. We are working on this and also welcome community contributions to help us.

[3]Upon contacting the authors, this version is experimental and used some different data than Vicuna 7B and 13B.

employed as a straightforward method to enhance arithmetic intensity and escape memory-bound limitations. However, with LLMs, both model parameters and the Key-Value (KV) cache consume substantial accelerator memory, hindering the utilization of large batch sizes. Existing methods to tackle this problem can be conceptually divided into two main categories: (1) Reducing memory consumption, thereby minimizing memory transfer overhead and enabling larger batch sizes, and (2) Minimizing the number of decoding steps to decrease latency directly.

Reducing KV Cache. Methods such as Multi-query attention [Shazeer, 2019] and Grouped-query attention [Ainslie et al., 2023] adopt a direct approach to diminish the KV cache. By utilizing fewer key and value heads in the attention modules relative to query heads, these strategies substantially cut the KV's memory consumption, thereby facilitating larger batch sizes and enhanced accelerator utilization [Pope et al., 2022]. Additionally, Zhang et al. [2023] proposes to selectively retain the most critical KV tokens, further reducing the KV cache. From a system perspective, Kwon et al. [2023] introduces a paged memory management scheme for reducing fragmentation of the KV cache.

Quantization. Quantization techniques are extensively used to shrink LLMs' memory consumption. Xiao et al. [2023a] apply rescaling between activations and parameters to eliminate outliers and simplify the quantization process. Dettmers et al. [2022] breaks down matrix multiplications into predominantly 8-bit and a minority of 16-bit operations. Frantar et al. [2022] iteratively round weight columns into 3/4 bits, while Lin et al. [2023] present an activation-aware quantization scheme to protect salient weights and compress LLMs to 3/4 bits. Kim et al. [2023] introduce a sparse plus low-precision pattern to handle a minor portion of vital weights, among other techniques.

use a smaller draft model to predict future words

Speculative Decoding. As an approach orthogonal to the aforementioned methods, speculative decoding [Leviathan et al., 2022, Chen et al., 2023] aims to execute several decoding steps in parallel, thus reducing the total number of steps required. This parallelization is realized by employing a smaller draft model to conjecture several subsequent words, which the LLMs then collectively evaluate and accept as appropriate. While resonating with non-autoregressive generation literature [Xiao et al., 2023b], this method is specifically tailored for LLMs to address the aforementioned inefficiency. Unlike previous works, we propose leveraging the original model to make predictions rather than introducing an additional draft model. This approach is more straightforward and seamlessly integrates into existing systems without the complexities of managing two models. Independently, Miao et al. [2023], Spector and Re [2023] propose the use of tree-structured attention to generate multiple candidates in parallel, where Miao et al. [2023] suggest employing an ensemble of models to propose candidates, and Spector and Re [2023] advocate adding another hierarchy for the draft model. After the first release of MEDUSA, we have seen many new works improving speculative decoding from the perspective of distillation [Liu et al., 2023, Zhou et al., 2023], making draft model training-free [He et al., 2023, Fu et al., 2023].

2.2 Sampling Scheme

The manner in which text is sampled from Large Language Models (LLMs) can significantly influence the quality of the generated output. Recent studies have revealed that direct sampling from a language model may lead to incoherent or nonsensical results [Pillutla et al., 2021, Holtzman et al., 2020]. In response to this challenge, *truncation sampling* schemes have been introduced [Fan et al., 2018, Basu et al., 2021, Meister et al., 2022, Hewitt et al., 2022, Meister et al., 2023]. These approaches aim to produce high-quality and diverse samples by performing sampling on a truncated distribution over a specific *allowed set* at each decoding step.

Different strategies define this allowed set in various ways. For example, top-k sampling [Fan et al., 2018] retains the k most likely words, whereas top-p sampling [Holtzman et al., 2020] incorporates the minimal set of words that account for p percent of the probability. Another method, known as typical decoding [Meister et al., 2023], employs the entropy of the predicted distribution to establish the threshold for inclusion. Hewitt et al. [2022] offers a unified framework to understand truncation sampling techniques comprehensively.

Drawing inspiration from these methods, our typical acceptance scheme aligns with the concept of defining an allowed set to exclude improbable candidates from the sampling process. However, we diverge because we do not insist on an exact correspondence between the output and language model

distribution. This deviation allows us to facilitate more diverse yet high-quality outputs, achieving greater efficiency without compromising the integrity of the generated text.

3 MEDUSA

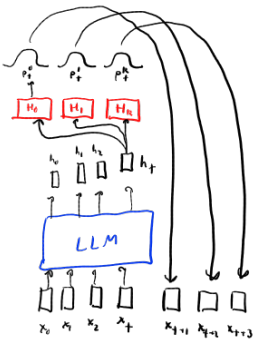
MEDUSA follows the same framework as speculative decoding, where each decoding step primarily consists of three substeps: (1) generating candidates, (2) processing candidates, and (3) accepting candidates. For MEDUSA, (1) is achieved by MEDUSA heads, (2) is realized by tree attention, and since MEDUSA heads are on top of the original model, the logits calculated in (2) can be used for substep (1) for the next decoding step. The final step (3) can be realized by either rejection sampling [Leviathan et al., 2022, Chen et al., 2023] or typical acceptance (Section 3.3.1). The overall pipeline is illustrated in Figure 1.

In this section, we first introduce the key components of MEDUSA, including MEDUSA heads, and tree attention. Then, we present two levels of fine-tuning procedures for MEDUSA to meet the needs of different use cases. Finally, we propose two extensions to MEDUSA, including self-distillation and typical acceptance, to handle situations where no training data is available for MEDUSA and to improve the efficiency of the decoding process, respectively.

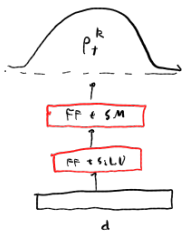
3.1 Key Components

3.1.1 MEDUSA Heads

In speculative decoding, subsequent tokens are predicted by an auxiliary draft model. This draft model must be small yet effective enough to generate continuations that the original model will accept. Fulfilling these requirements is a challenging task, and existing approaches [Spector and Re, 2023, Miao et al., 2023] often resort to separately *pre-training* a smaller model. This pre-training process demands substantial additional computational resources. For example, in [Miao et al., 2023], a reported 275 NVIDIA A100 GPU hours were used. Additionally, separate pre-training can potentially create a distribution shift between the draft model and the original model, leading to continuations that the original model may not favor. Chen et al. [2023] have also highlighted the complexities of serving multiple models in a distributed environment.



To streamline and democratize the acceleration of LLM inference, we take inspiration from Stern et al. [2018] and introduce MEDUSA heads. These are additional decoding heads appended to the last hidden states of the original model. Specifically, given the original model's last hidden states h_t at position t, we add K decoding heads to h_t. The k-th head is used to predict the token in the $(t+k+1)$-th position of the next tokens (the original language model head is used to predict the $(t+1)$-th position). The prediction of the k-th head is denoted as $p_t^{(k)}$, representing a distribution over the vocabulary, while the prediction of the original model is denoted as $p_t^{(0)}$. Following the approach of Stern et al. [2018], we utilize a single layer of feed-forward network with a residual connection for each head. We find that this simple design is sufficient to achieve satisfactory performance. The definition of the k-th head is outlined as:



$$p_t^{(k)} = \text{softmax}\left(W_2^{(k)} \cdot \left(\text{SiLU}(W_1^{(k)} \cdot h_t) + h_t\right)\right), \text{ where } W_2^{(k)} \in \mathbb{R}^{d \times V}, W_1^{(k)} \in \mathbb{R}^{d \times d}.$$

We initialize $W_1^{(k)}$ identically to the original language model head, and $W_2^{(k)}$ to zero. This aligns the initial prediction of MEDUSA heads with that of the original model. The SiLU activation function [Elfwing et al., 2017] is employed following the Llama models [Touvron et al., 2023].

Unlike a draft model, MEDUSA heads are trained in conjunction with the original backbone model, which can remain *frozen* during training (MEDUSA-1) or be trained together (MEDUSA-2). This method allows for fine-tuning large models even on a single GPU, taking advantage of the powerful base model's learned representations. Furthermore, it ensures that the distribution of the MEDUSA heads aligns with that of the original model, thereby mitigating the distribution shift problem. Additionally, since the new heads consist of just a single layer akin to the original language model head, MEDUSA does not add complexity to the serving system design and is friendly to distributed settings. We will discuss the training recipe for MEDUSA heads in Section 3.2.

3.1.2 Tree Attention

Through MEDUSA heads, we obtain probability predictions for the subsequent $K+1$ tokens. These predictions enable us to create length-$K+1$ continuations as candidates. While the speculative decoding studies [Leviathan et al., 2022, Chen et al., 2023] suggest sampling a single continuation as the candidate, leveraging multiple candidates during decoding can enhance the expected acceptance length within a decoding step. Nevertheless, more candidates can also raise computational demands. To strike a balance, we employ a tree-structured attention mechanism to process multiple candidates concurrently. This attention mechanism diverges from the traditional causal attention paradigm.

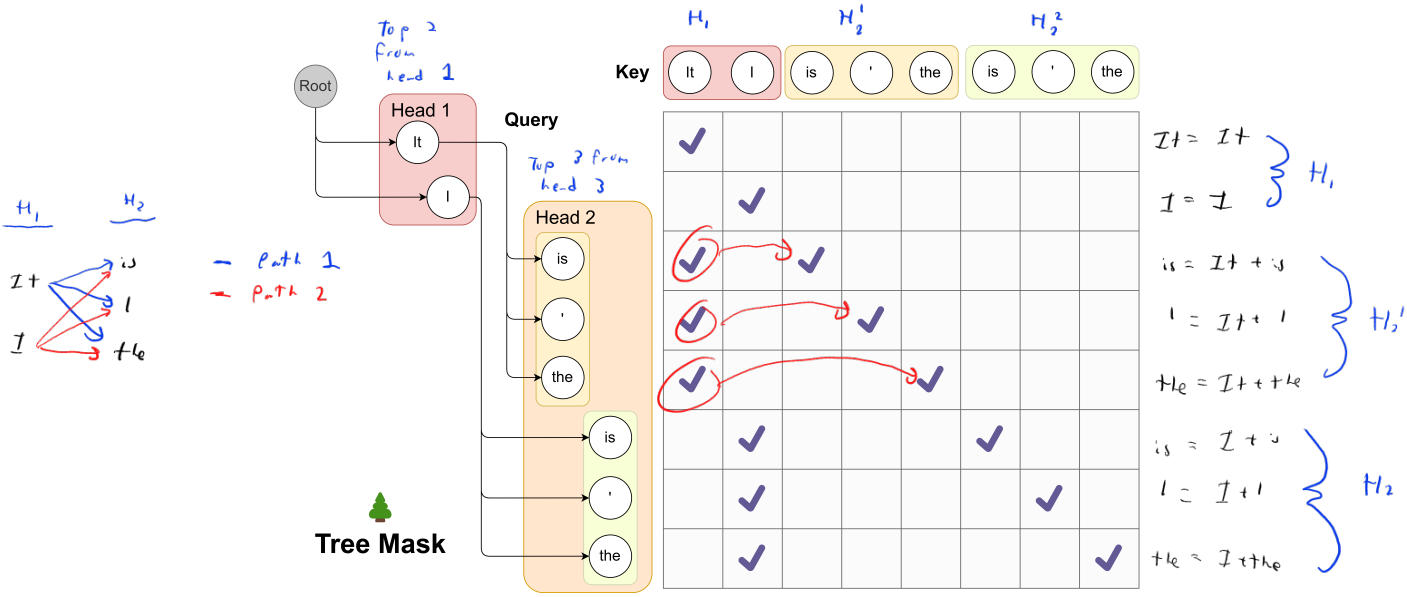


Figure 2: **Tree Attention Illustrated.** This visualization demonstrates the use of tree attention to process multiple candidates concurrently. As exemplified, the top-2 predictions from the first MEDUSA head and the top-3 from the second result in a total of $2 \times 3 = 6$ candidates. Each of these candidates corresponds to a distinct branch within the tree structure. To guarantee that each token only accesses its predecessors, we devise an attention mask that exclusively permits attention flow from the current token back to its antecedent tokens. The positional indices for positional encoding are adjusted in line with this structure.

Within this framework, only tokens from the same continuation are regarded as historical data. Drawing inspiration from the concept of embedding graph structures into attention as proposed in the graph neural network domain [Ying et al., 2021], we incorporate the tree structure into our attention mask, visualized in Figure 2. For a given k-th head, its top-s_k predictions serve as the basis for candidate formation, where s_k is a designated hyperparameter. These candidates are established by determining the Cartesian product of the top-s_k predictions from each head. For instance, in Figure 2, with $s_1 = 2$ and $s_2 = 3$, each first head prediction can be succeeded by any prediction from the second head. This leads to a tree structure where s_k branches exist at the k-th level (considering a virtual root as the 0-level, in practice, this 0-level is for the prediction of the language model head of the original model, which can be sampled independently). Within this tree, only a token's predecessors are seen as historical context, and our attention mask ensures that the attention is only applied on a token's predecessors. By employing this mask and properly setting the positional indices for positional encoding, we can process numerous candidates simultaneously without the need to expand the batch size. The cumulative number of new tokens is calculated as $\sum_{k=1}^{K} \prod_{i=1}^{k} s_i$.

In this section, we demonstrate the most simple and regular way to construct the tree structure by taking the Cartesian product. However, it is possible to construct the tree structure in a more sophisticated way and exploit the unbalanced accuracy of different top predictions of different heads. We will discuss this in Section 3.3.3.

3.2 Training Strategies

At the most basic level, we can train MEDUSA heads by freezing the backbone model and focusing solely on the MEDUSA heads. This approach is straightforward and requires minimal computational resources. However, training the backbone in conjunction with the MEDUSA heads can significantly enhance the accuracy of the MEDUSA heads. Depending on the computational resources and the specific requirements of the use case, we propose two levels of training strategies for MEDUSA heads.

In this section, we assume the availability of a training dataset that aligns with the target model's output distribution. This could be the dataset used for Supervised Fine-Tuning (SFT) of the target model. We will discuss how to eliminate the need for such a dataset using a self-distillation approach in Section 3.3.2.

3.2.1 MEDUSA-1: Frozen Backbone

To train MEDUSA heads with a frozen backbone model, we can use the cross-entropy loss between the prediction of MEDUSA heads and the ground truth. Specifically, given the ground truth token y_{t+k+1} at position $t+k+1$, the loss for the k-th head is $\mathcal{L}_k = -\log p_t^{(k)}(y_{t+k+1})$ where $p_t^{(k)}(y)$ denotes the probability of token y predicted by the k-th head. We also observe that $\mathcal{L}_k$ is larger when k is larger, which is reasonable since the prediction of the k-th head is more uncertain when k is larger. Therefore, we can add a weight λ_k to $\mathcal{L}_k$ to balance the loss of different heads. And the total MEDUSA loss is:

$$\mathcal{L}_{\text{MEDUSA-1}} = \sum_{k=1}^{K} -\lambda_k \log p_t^{(k)}(y_{t+k+1}). \tag{1}$$

In practice, we set λ_k as the k-th power of a constant like 0.8. Since we only use the backbone model for providing the hidden states, we can use a quantized version of the backbone model to reduce the memory consumption. This introduces a more democratized way to accelerate LLM inference, as with the quantization, MEDUSA can be trained for a large model on a single consumer GPU similar to QLoRA [Dettmers et al., 2023]. The training only takes a few hours (e.g., 5 hours for MEDUSA-1 on Vicuna 7B model with a single NVIDIA A100 PCIE GPU to train on 60k ShareGPT samples).

3.2.2 MEDUSA-2: Joint Training

To further improve the accuracy of MEDUSA heads, we can train MEDUSA heads together with the backbone model. However, this requires a special training recipe to preserve the backbone model's next-token prediction capability and output quality. To achieve this, we propose three strategies:

- **Combined loss**: To keep the backbone model's next-token prediction capability, we need to add the cross-entropy loss of the backbone model $\mathcal{L}_{\text{LM}} = -\log p_t^{(0)}(y_{t+1})$ to the MEDUSA loss. We also add a weight λ_0 to balance the loss of the backbone model and the MEDUSA heads. Therefore, the total loss is:

$$\mathcal{L}_{\text{MEDUSA-2}} = \mathcal{L}_{\text{LM}} + \lambda_0 \mathcal{L}_{\text{MEDUSA-1}}. \tag{2}$$

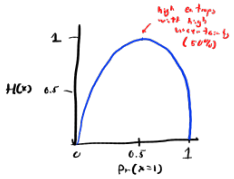
- **Differential learning rates**: Since the backbone model is already well-trained and the MEDUSA heads need more training, we can use separate learning rates for them to enable faster convergence of MEDUSA heads while preserving the backbone model's capability.
- **Heads warmup**: Noticing that at the beginning of training, the MEDUSA heads have a large loss, which leads to a large gradient and may distort the backbone model's parameters. Following the idea from Kumar et al. [2022], we can employ a two-stage training process. In the first stage, we only train the backbone model as MEDUSA-1. In the second stage, we train the backbone model and MEDUSA heads together with a warmup strategy. Specifically, we first train the backbone model for a few epochs, then train the MEDUSA heads together with the backbone model. Besides this simple strategy, we can also use a more sophisticated warmup strategy by gradually increasing the weight λ_0 of the backbone model's loss. We find both strategies work well in practice.

Putting these strategies together, we can train MEDUSA heads together with the backbone model without hurting the backbone model's capability. Moreover, this recipe can be applied together with Supervised Fine-Tuning (SFT), enabling us to get a model with native MEDUSA support.

3.3 Extensions

3.3.1 Typical Acceptance

In speculative decoding papers [Leviathan et al., 2022, Chen et al., 2023], authors employ rejection sampling to yield diverse outputs that align with the distribution of the original model. However, subsequent implementations [Joao Gante, 2023, Spector and Re, 2023] reveal that this sampling strategy results in diminished efficiency as the sampling temperature increases. Intuitively, this can be comprehended in the extreme instance where the draft model is the same as the original one. Here, when using greedy decoding, all output of the draft model will be accepted, therefore maximizing the efficiency. Conversely, rejection sampling introduces extra overhead, as the draft model and the original model are sampled independently. Even if their distributions align perfectly, the output of the draft model may still be rejected.



$H = -\sum_{i=1}^{n} p_i \log_2 p_i$

However, in real-world scenarios, sampling from language models is often employed to generate diverse responses, and the temperature parameter is used merely to modulate the "creativity" of the response. Therefore, higher temperatures should result in more opportunities for the original model to accept the draft model's output. We ascertain that it is typically unnecessary to match the distribution of the original model. Thus, we propose employing a *typical acceptance* scheme to select plausible candidates rather than using rejection sampling. This approach draws inspiration from truncation sampling studies [Hewitt et al., 2022] (refer to Section 2 for an in-depth explanation). Our objective is to choose candidates that are *typical*, meaning they are not exceedingly improbable to be produced by the original model. We use the prediction probability from the *original model* as a natural gauge for this and establish a threshold based on the prediction distribution to determine acceptance. Specifically, given $x_1, x_2, \cdots, x_n$ as context, when evaluating the candidate sequence $(x_{n+1}, x_{n+2}, \cdots, x_{n+T+1})$ (composed by top predictions of the original language model head and MEDUSA heads), we consider the condition

used to select sequences that are "typical", that is the prob of the sequence is the entropy of the sequence is not too high or too

$$p_{\text{original}}(\overbrace{x_{n+k}}^{\text{prob of next n tokens}} | x_1, x_2, \cdots, x_{n+k-1}) > \min\left(\epsilon, \delta \exp\left(-\overbrace{H(p_{\text{original}}(\cdot | x_1, x_2, \cdots, x_{n+k-1}))}^{\text{entropy of sequence with new tokens}}\right)\right), \quad \text{the higher the sequence entropy, the lower the value}$$

where $H(\cdot)$ denotes the entropy function, and ϵ, δ are hyperparameters. This criterion is adapted from Hewitt et al. [2022] and rests on two observations: (1) tokens with relatively high probability are meaningful, and (2) when the distribution's entropy is high, various continuations may be deemed reasonable. During decoding, every candidate is evaluated using this criterion, and a *prefix* of the candidate is accepted if it satisfies the condition. To guarantee the generation of at least one token at each step, we apply *greedy decoding* for the first token and *unconditionally* accept it while employing typical acceptance for subsequent tokens. The final prediction for the current step is determined by the *longest accepted prefix* among all candidates.

Examining this scheme leads to several insights. Firstly, when the temperature is set to 0, it reverts to greedy decoding, as only the most probable token possesses non-zero probability. As the temperature surpasses 0, the outcome of greedy decoding will consistently be accepted with appropriate ϵ, δ, since those tokens have the maximum probability, yielding maximal speedup. Likewise, in general scenarios, an increased temperature will correspondingly result in longer accepted sequences, as corroborated by our experimental findings.

3.3.2 Self-Distillation

Assume distribution of new data is the same as what the model was trained on, but it may not be so

In Section 3.2, we assume the existence of a training dataset that matches the target model's output distribution. However, this is not always the case. For example, the model owners may only release the model without the training data, or the model may have gone through a Reinforcement Learning with Human Feedback (RLHF) procedure, which makes the output distribution of the model different from the training dataset. To tackle this issue, we propose an automated self-distillation pipeline to use the model itself to generate the training dataset for MEDUSA heads, which matches the output distribution of the model.

Steps:
1. Find dataset with similar distribution
2. Take data from dataset and ask model to reply
- This creates an "in distribution" dataset for the model

The dataset generation process is straightforward. We first take a public seed dataset from a domain similar to the target model; for example, using the ShareGPT [ShareGPT, 2023] dataset for chat models. Then, we simply take the prompts from the dataset and ask the model to reply to the prompts. In order to obtain multi-turn conversation samples, we can sequentially feed the prompts from the seed dataset to the model. Or, for models like Zephyr 7B [Tunstall et al., 2023], which are trained on both roles of the conversation, they have the ability to self-talk, and we can simply feed the first prompt and let the model generate multiple rounds of conversation.

For MEDUSA-1, this dataset is sufficient for training MEDUSA heads. However, for MEDUSA-2, we observe that solely using this dataset for training the backbone and MEDUSA heads usually leads to a lower generation quality. In fact, even without training MEDUSA heads, training the backbone model with this dataset will lead to performance degradation. This suggests that we also need to use the original model's probability prediction instead of using the ground truth token as the label for the backbone model, similar to classic knowledge distillation works [Kim and Rush, 2016]. Concretely, the loss for the backbone model is:

$$\mathcal{L}_{\text{LM-distill}} = KL(p^{(0)}_{\text{original},t} || p^{(0)}_{t}),$$

where $p^{(0)}_{\text{original},t}$ denotes the probability distribution of the original model's prediction at position t.

However, naively, to obtain the original model's probability prediction, we need to maintain two models during training, increasing the memory requirements. To further alleviate this issue, we propose a simple yet effective way to exploit the self-distillation setup. We can use a parameter-efficient adapter like LoRA [Hu et al., 2021] for fine-tuning the backbone model. In this way, the original model is simply the model with the adapter turned off. Therefore, the distillation does not require additional memory consumption. Together, this self-distillation pipeline can be used to train MEDUSA-2 without hurting the backbone model's capability and introduce almost no additional memory consumption. Lastly, one tip about using self-distillation is that it is preferable to use LoRA without quantization in this case, otherwise, the teacher model will be the quantized model, which may lead to a lower generation quality.

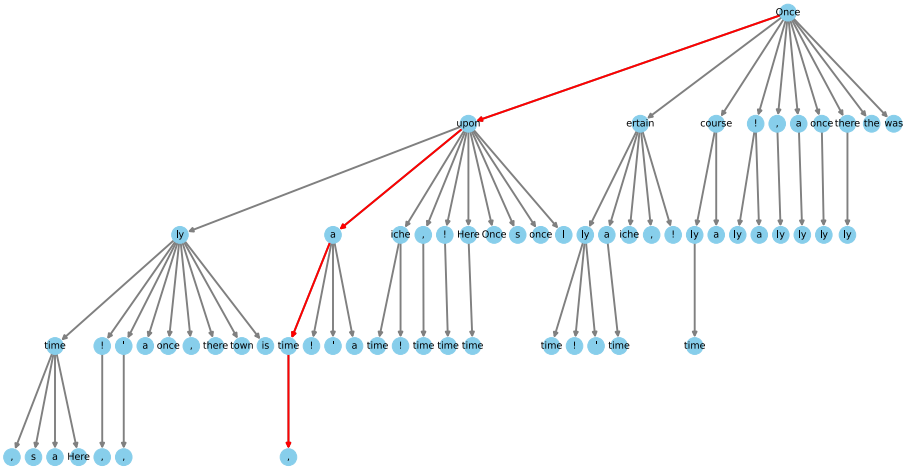


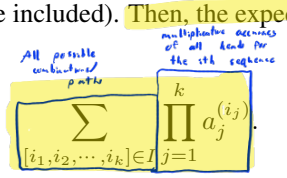
Figure 3: Visualization of a sparse tree setting for MEDUSA-2 Vicuna-7B. The tree has depth 4 which indicates 4 MEDUSA heads involved in calculation. Each node indicates a token from a top-k prediction of a MEDUSA head, and the edges show the connections between them. The red lines highlight the path that correctly predicts the future tokens.

3.3.3 Searching for the Optimized Tree Construction

In Section 3.1.2, we present the simplest way to construct the tree structure by taking the Cartesian product. However, with a fixed number of total nodes in the tree, a regular tree structure may not be the best choice. Intuitively, those candidates composed of the top predictions of different heads may have different accuracies. Therefore, we can leverage an estimation of the accuracy to construct the tree structure.

Specifically, we can use a calibration dataset and calculate the accuracies of the top predictions of different heads. Let $a_k^{(i)}$ denote the accuracy of the i-th top prediction of the k-th head. Assuming the accuracies are independent, we can estimate the accuracy of a candidate sequence composed by the top $[i_1, i_2, \cdots, i_k]$ predictions of different heads as $\prod_{j=1}^{k} a_j^{(i_j)}$. Let I denote the set of all possible combinations of $[i_1, i_2, \cdots, i_k]$ and each element of I can be mapped to a node of the tree

(not only leaf nodes but all nodes are included). Then, the expectation of the acceptance length of a candidate sequence is:



$$\sum_{[i_1,i_2,\cdots,i_k]\in I} \prod_{j=1}^{k} a_j^{(i_j)}.$$

Thinking about building a tree by adding nodes one by one, the contribution of a new node to the expectation is exactly the accuracy associated with the node. Therefore, we can greedily add nodes to the tree by choosing the node that is connected to the current tree and has the highest accuracy. This process can be repeated until the total number of nodes reaches the desired number. In this way, we can construct a tree structure that maximizes the expectation of the acceptance length.

Fig. 3 illustrates the structure of a sparsely constructed tree for the MEDUSA-2 Vicuna-7B model. This tree structure extends four levels deep, indicating the engagement of four MEDUSA heads in the computation. The tree is initially formed through a Cartesian product approach and subsequently refined by pruning based on the statistical expectations of the top-k predictions from each MEDUSA head measured on the Alpaca-eval dataset Dubois et al. [2023]. The tree's lean towards the left visually represents the algorithm's preference for nodes with higher probabilities on each head.

4 Experiments

In this section, we present two sets of experiments to demonstrate the effectiveness of MEDUSA under different settings. First, we evaluate MEDUSA on the Vicuna-7B and 13B models [Chiang et al., 2023] to show the performance of MEDUSA-1 and MEDUSA-2. Second, we evaluate MEDUSA on the Vicuna-33B and Zephyr-7B [Tunstall et al., 2023] models to study the effectiveness of self-distillation because for Vicuna-33B model, the training dataset is not publicly available, and for Zephyr-7B model, the model is trained with RLHF.

We clarify three commonly used terms: a) Acceleration rate: This refers to the average number of tokens decoded per decoding step. In a standard auto-regressive model, this rate is 1.0. b) Overhead: This is used to characterize the per decoding step overhead compared to classic decoding, and is calculated by dividing the average per step latency of the MEDUSA models by that of the vanilla model. c) Speedup: This refers to the wall-time acceleration rate. Following these definitions, we have the relation: Speedup = Acceleration rate / Overhead.

4.0.1 Shared Settings

5 heads
0.8^k for head decay

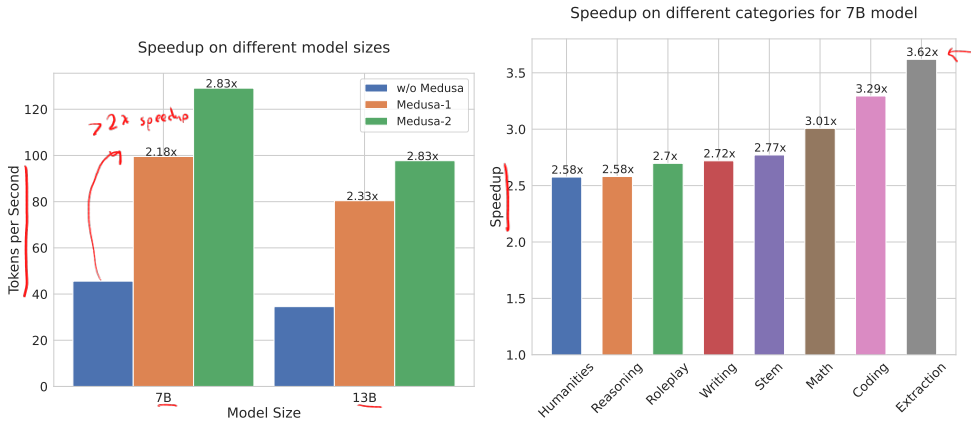
LoRA applied to all linear layers

For all the experiments, we use the Axolotl [Axolotl, 2023] framework for training. We use a cosine learning rate scheduler with warmup and use 8-bit AdamW [Dettmers et al., 2021] optimizer. We train 5 MEDUSA heads with 1 layer and set λ_k in Eq. (1) to be 0.8^k. For MEDUSA-2, we use either LoRA [Hu et al., 2021] or QLoRA [Dettmers et al., 2023] for fine-tuning and set the learning rate of MEDUSA heads to be 4 times larger than the backbone model. LoRA is applied to all the linear layers of the backbone model, including the language model head. The rank of LoRA adapter is set to 32, and α is set to 16. A dropout of 0.05 is added to the LoRA adapter.

4.1 Case Study: MEDUSA-1 v.s. MEDUSA-2 on Vicuna 7B and 13B

4.1.1 Experimental Setup

We use the Vicuna model class [Chiang et al., 2023], which encompasses chat models of varying sizes (7B, 13B, 33B) that are fine-tuned from the Llama model [Touvron et al., 2023]. Among them, the 7B and 13B models are trained on the ShareGPT [ShareGPT, 2023] dataset, while the 33B model is an experimental model and is trained on a private dataset. In this section, we use the ShareGPT dataset to train the MEDUSA heads on the 7B and 13B models for 2 epochs. We use the v1.5 version of Vicuna models, which are fine-tuned from Llama-2 models with sequence length 4096. We use a global batch size of 64 and a peak learning rate of $5e^{-4}$ for the backbone and $2e^{-3}$ for MEDUSA heads and warmup for 40 steps. We use 4-bit quantized backbone models for both models. We first train the models with MEDUSA-1 and use these trained models as initialization to train MEDUSA-2. We employ QLoRA for MEDUSA-2 and the λ_0 in Eq. (2) is set to be 0.2.



(a) (b)

Figure 4: Left: Speed comparison of baseline, MEDUSA-1 and MEDUSA-2 on Vicuna-7B/13B. MEDUSA-1 achieves more than 2× wall-time speedup compared to the baseline implementation while MEDUSA-2 further improves the speedup by a significant margin. Right: Detailed speedup performance of Vicuna-7B on 8 categories from MT-Bench.

4.1.2 Results

We collect the results and show them in Fig. 4. The baseline is the vanilla Huggingface implementation. In Fig. 4a, we can see that for the 7B models, MEDUSA-1 and MEDUSA-2 configurations lead to a significant increase in speed, measuring in tokens processed per second. MEDUSA-1 shows a 2.18× speedup, while MEDUSA-2 further improves this to a 2.83×. When applied to the larger 13B model, MEDUSA-1 results in a 2.33× speed increase, while MEDUSA-2 maintains a similar performance gain of 2.83× over the baseline. We also plot the speedup per category for MEDUSA-2 Vicuna-7B model. We observe that the "Coding" category benefits from a 3.29× speedup, suggesting that MEDUSA is particularly effective for tasks in this domain. This points to a significant potential for optimizing coding LLMs, widely used in software development and other programming-related tasks. The "Extraction" category shows the highest speedup at 3.62×, indicating that this task is highly optimized by the MEDUSA. Overall, the results suggest that the MEDUSA significantly enhances inference speed across different model sizes and tasks.

4.2 Case Study: Training with Self-Distillation on Vicuna-33B and Zephyr-7B

4.2.1 Experimental Setup

In this case study, we focus on the cases where self-distillation is needed. We use the Vicuna-33B model [Chiang et al., 2023] and the Zephyr-7B model [Tunstall et al., 2023] as examples. Following the procedure described in Section 3.3.2, we first generate the datasets with some seed prompts. We use ShareGPT [ShareGPT, 2023] and UltraChat [Ding et al., 2023] as the seed datasets and collect a dataset at about $100k$ samples for both cases. Interestingly, we find that the Zephyr model can continue to generate multiple rounds of conversation with a single prompt, which makes it easy to collect a large dataset. For Vicuna-33B, we generate the multi-turn conversations by iteratively feeding the prompts from each multi-turn seed conversation. Both models are trained with sequence length 2048 and batch size 128. We use MEDUSA-2 for both models and instead of using a two-stage training procedure, we use a sine schedule for the θ_0 to gradually increase the value to its peak at the end of the training, we find this approach is equally effective. We set the peak learning rate of the backbone LoRA adapter to be $1e^{-4}$ and the warmup steps to be 20. Since the self-distillation loss is relatively small, we set the λ_0 in Eq. (2) to be 0.01.

4.2.2 Results

Table 1 complements these findings by comparing various MEDUSA-2 models in terms of their acceleration rate, overhead, and quality on MT-Bench. Notably, while the MEDUSA-2 Vicuna-33B

model shows a lower acceleration rate, it maintains a comparable quality. We hypothesize that this is due to a mismatch between the hidden training dataset and the dataset we used for self-distillation.

These results underscore the complex interplay between speed and performance when scaling up model sizes and applying self-distillation techniques. The findings also highlight the potential of the MEDUSA-2 configuration to boost efficiency in processing while carefully preserving the quality of the model's outputs, suggesting a promising direction for co-optimizing LLMs with MEDUSA heads.

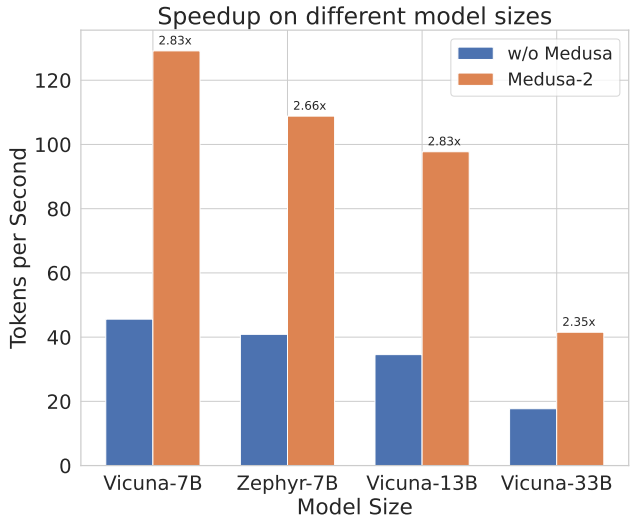


Figure 5: Speedup of various models with MEDUSA-2. MEDUSA-2 shows significant speed improvement over all the models, while models trained with self-distillation have weaker speedup due to the trade-off between preserving model quality and boosting model speed.

Model Name	Vicuna-7B	Zephyr-7B	Vicuna-13B	Vicuna-33B
Acc. rate	3.47	3.14	3.51	3.01
Overhead	1.22	1.18	1.23	1.27
Quality	6.18 (+0.01)	7.25 (-0.07)	6.43 (-0.14)	7.18 (+0.05)

Table 1: Comparison of various MEDUSA-2 models. The quality denotes the average scores on the MT-Bench benchmark [Zheng et al., 2023]. MEDUSA-2 achieves promising acceleration rate with mild overhead and preserves the model quality.

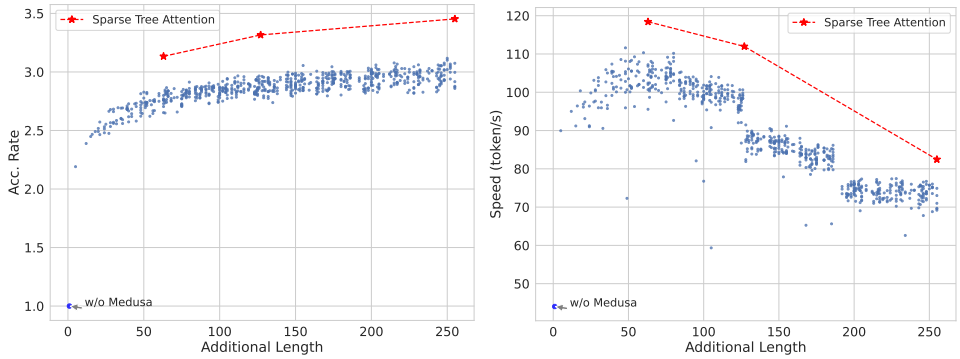
4.3 Ablation Study

4.3.1 Configuration of Tree Attention

The ablation study of tree attention is conducted on the writing and roleplay categories from the MT-Bench dataset [Zheng et al., 2023] using MEDUSA-2 Vicuna-7B. We target to depict tree attention's motivation and its performance.

Fig. 6a compares the acceleration rate of randomly sampled dense tree configurations (Section. 3.1.2, depicted by blue dots) against optimized sparse tree settings (Section. 3.3.3, shown with red stars). The sparse tree configuration with 64 nodes shows a better acceleration rate than the dense tree settings with 256 nodes. Fig. 6b presents the speed for both dense and sparse tree settings. The trend observed here indicates a notable decrease in speed as the additional length increases. This suggests that while sparse tree attention is beneficial for maintaining a steady acceleration rate, it comes with the trade-off of reduced speed, particularly at higher additional lengths.

The observed decline in speed is attributed to the increased overhead introduced by the hardware architecture. While a more complex tree can improve the acceleration rate, it does so at the cost of speed due to the hardware-imposed overhead.



(a) (b)

Figure 6: Evaluation of additional length introduced by trees. Left: The acceleration rate for randomly sampled dense tree settings (blue dots) and optimized sparse tree settings (red stars). Right: The speed (tokens/s) for both settings. The trend lines indicate that while the acceptance rate remains relatively stable for sparse trees, there is a notable decrease in speed as the additional length increases.

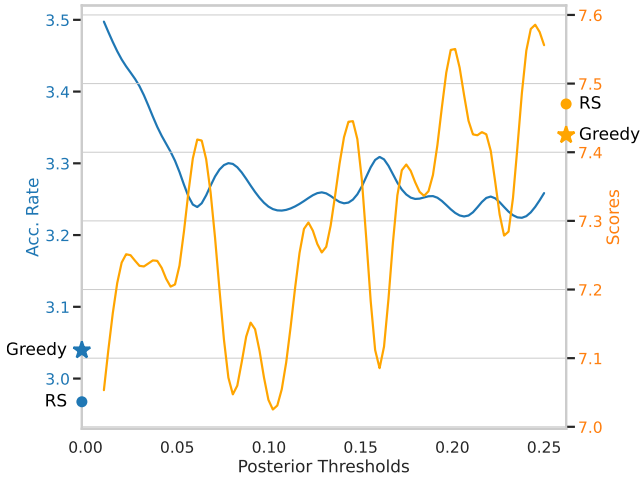


Figure 7: Performance comparison of MEDUSA using proposed typical sampling. The plot illustrates the acceleration rate (Acc. Rate) and averaged scores on the "Writing" and "Roleplay" (MT benchmark) with a fixed temperature of 0.7 for 3 different settings: greedy sampling with MEDUSA, random sampling, and typical sampling under different thresholds. The model is fully fine-tuned Vicuna-7B.

4.3.2 Thresholds of Typical Acceptance

The thresholds of typical acceptance are studied on the "Writing" and "Roleplay" categories from the MT-Bench dataset [Zheng et al., 2023] using MEDUSA-2 Vicuna 7B. Utilizing the Vicuna 7B model, we aligned our methodology with the approach delineated by Hewitt et al. [2022] setting the $\alpha = \sqrt{\epsilon}$. Fig. 7 presents a comparative analysis of our model's performance across various sampling settings. These settings range from a threshold ϵ starting at 0.01 and incrementally increasing to 0.25 in steps of 0.01. Our observations indicate a discernible trade-off: as ϵ increases, there is an elevation in quality at the expense of a reduced acceleration rate. Furthermore, for tasks demanding creativity, it is noted that the default random sampling surpasses greedy sampling in performance, and the proposed typical sampling is comparable with random sampling when ϵ increases.

	Baseline	Direct Fine-tuning	MEDUSA-1	MEDUSA-2
Quality	6.17	5.925	6.23	6.18
Speed Up	N/A	N/A	2.18	2.83

Table 2: Comparison of Different Settings Vicuna-7B. Quality is obtained by evaluating models on MT-Bench.

4.3.3 Effectiveness of Two-stage Fine-tuning

We examine the performance differences between two fine-tuning strategies for the Vicuna-7B model in Table 2. We provided the comparison of directly fine-tuning the model with the MEDUSA heads vs. MEDUSA-2 that involves two-stage fine-tuning described in Section 3.2.2. The findings indicate that implementing our MEDUSA-2 for fine-tuning maintains the model's quality and concurrently improves the speedup versus MEDUSA-1.

4.4 Discussion

In conclusion, we have proposed MEDUSA, a novel method to accelerate large language model inference by equipping models with additional predictive decoding heads. MEDUSA allows models to generate multiple tokens per step, overcoming the bottleneck of sequential auto-regressive decoding. We have demonstrated two procedures, MEDUSA-1 and MEDUSA-2, for efficiently training these extra heads while preserving model performance. Experiments on models of varying sizes and training methods show consistent speedups of 2.3-3.6$\times$ on single prompt inference across different prompt types and models.

Key advantages of MEDUSA include its simplicity, parameter efficiency, and ease of integration into existing systems. By building on top of speculative decoding concepts, MEDUSA avoids the need for specialized draft models. The typical acceptance scheme also removes complications from rejection sampling while still providing reasonable outputs. Finally, the fine-tuning procedures ensure high-quality generations without affecting the performance of the original model.

Acknowledgements

We extend our heartfelt gratitude to several individuals whose contributions were invaluable to this project:

- Zhuohan Li, for his invaluable insights on LLM serving. If you haven't already, do check out Zhuohan's vLLM project—it's nothing short of impressive.
- Shaojie Bai, for engaging in crucial discussions that helped shape the early phases of this work.
- Denny Zhou, for introducing the truncation sampling scheme to Tianle and encouraging Tianle to explore the area of LLM serving.
- Yanping Huang, for pointing out the memory-bound challenges associated with LLM serving to Tianle.

References

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

Axolotl. Axolotl. `https://github.com/OpenAccess-AI-Collective/axolotl`, 2023.

Sourya Basu, Govardana Sachitanandam Ramachandran, Nitish Shirish Keskar, and Lav R. Varshney. {MIROSTAT}: A {neural} {text} {decoding} {algorithm} {that} {directly} {controls} {perplexity}. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=W1G1JZEIy5_`.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. February 2023. doi: 10.48550/ARXIV.2302.01318.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023. URL `https://lmsys.org/blog/2023-03-30-vicuna/`.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Tim Dettmers, M. Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit optimizers via block-wise quantization. *International Conference on Learning Representations*, 2021.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations, 2023.

Yann Dubois, Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpacafarm: A simulation framework for methods that learn from human feedback, 2023.

Stefan Elfwing, E. Uchibe, and K. Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 2017. doi: 10.1016/j.neunet.2017.12.012.

Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2018. doi: 10.18653/v1/p18-1082.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Breaking the sequential dependency of llm inference using lookahead decoding, November 2023. URL `https://lmsys.org/blog/2023-11-21-lookahead-decoding/`.

Google. Palm 2 technical report, 2023. URL `https://ai.google/static/documents/palm2techreport.pdf`.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D Lee, and Di He. Rest: Retrieval-based speculative decoding. *arXiv preprint arXiv: 2311.08252*, 2023.

John Hewitt, Christopher D. Manning, and Percy Liang. Truncation sampling as language model desmoothing. October 2022. doi: 10.48550/ARXIV.2210.15191.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rygGQyrFvH.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *ICLR*, 2021.

Joao Gante. Assisted generation: a new direction toward low-latency text generation, 2023. URL https://huggingface.co/blog/assisted-generation.

Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. Squeezellm: Dense-and-sparse quantization. *arXiv preprint arXiv:2306.07629*, 2023.

Yoon Kim and Alexander M. Rush. Sequence-level knowledge distillation. *EMNLP*, 2016.

Ananya Kumar, Aditi Raghunathan, Robbie Jones, Tengyu Ma, and Percy Liang. Fine-tuning can distort pretrained features and underperform out-of-distribution. *International Conference on Learning Representations*, 2022.

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. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. November 2022. doi: 10.48550/ARXIV.2211.17192.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Ion Stoica, Zhijie Deng, Alvin Cheung, and Hao Zhang. Online speculative decoding. *arXiv preprint arXiv: 2310.07177*, 2023.

Clara Meister, Gian Wiher, Tiago Pimentel, and Ryan Cotterell. On the probability-quality paradox in language generation. March 2022. doi: 10.48550/ARXIV.2203.17217.

Clara Meister, Tiago Pimentel, Gian Wiher, and Ryan Cotterell. Locally typical sampling. *Transactions of the Association for Computational Linguistics*, 11:102–121, 2023.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Rae Ying Yee Wong, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. Specinfer: Accelerating generative llm serving with speculative inference and token tree verification. *arXiv preprint arXiv:2305.09781*, 2023.

OpenAI. Gpt-4 technical report, 2023.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

Krishna Pillutla, Swabha Swayamdipta, Rowan Zellers, John Thickstun, Sean Welleck, Yejin Choi, and Zaid Harchaoui. MAUVE: Measuring the gap between neural text and human text using divergence frontiers. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=Tqx7nJp7PR.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. November 2022. doi: 10.48550/ARXIV.2211.05102.

ShareGPT. ShareGPT. https://huggingface.co/datasets/Aeala/ShareGPT_Vicuna_unfiltered, 2023.

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

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

Mitchell Stern, Noam M. Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. *Neural Information Processing Systems*, 2018.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Kashif Rasul, Younes Belkada, Shengyi Huang, Leandro von Werra, Clémentine Fourrier, Nathan Habib, Nathan Sarrazin, Omar Sanseviero, Alexander M. Rush, and Thomas Wolf. Zephyr: Direct distillation of lm alignment, 2023.

Heming Xia, Tao Ge, Si-Qing Chen, Furu Wei, and Zhifang Sui. Speculative decoding: Lossless speedup of autoregressive translation, 2023. URL https://openreview.net/forum?id=H-VlwsYvVi.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR, 2023a.

Yisheng Xiao, Lijun Wu, Junliang Guo, Juntao Li, Min Zhang, Tao Qin, and Tie-yan Liu. A survey on non-autoregressive generation for neural machine translation and beyond. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023b.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34:28877–28888, 2021.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H _2 o: Heavy-hitter oracle for efficient generative inference of large language models. *arXiv preprint arXiv:2306.14048*, 2023.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. Distillspec: Improving speculative decoding via knowledge distillation. *arXiv preprint arXiv: 2310.08461*, 2023.