DRAFT MODEL GENERATION FOR SPECULATIVE DECODING

Arian Raje **Sushanth Gangireddy** **Victor Akinwande**

August 25, 2024

ABSTRACT

Autoregressive decoding using Large Language Models (LLMs) is slow as only a single token is generated at a time. To generate a sequence of n tokens, the model must be run n times with each generated token appended onto the prompt to generate the next token in the sequence. Speculative decoding offers an alternative approach to inference where a smaller and more efficient draft model is used to generate tokens autoregressively. Then the original LLM, which is referred to as the target model, verifies the tokens generated by the draft model in parallel, accepting and rejecting a portion of them. However, most speculative decoding research either assumes that a draft model with the same vocabulary and training data already exists for a given target model or focuses entirely on generating an effective draft model for a specific target model of interest. Our project focuses on draft model construction for an arbitrary target model. We use traditional model compression techniques and evaluate their effectiveness in a speculative decoding inference framework. We find that choosing the method that best balances model size and model utility leads to consistent speedups in inference.

1 Introduction

Large Language Models (LLMs) have emerged as the preeminent approach for natural language tasks in a variety of domains [1]. These models, which generally use Transformer architectures, have consistently demonstrated state-of-the-art performance in generating human-like language and text. However, LLMs have increased in size year after year, giving rise to a so-called "Moore's Law for LLMs" where the highest performing LLM is approximately 10x the size of the highest performing LLM from just the previous year [2]. This scale has resulted in serious computational bottlenecks for machine learning (ML) practitioners and researchers which has lead to a more serious discussion of improving the efficiency of LLMs. This is especially true in the context of LLM inference, where decoder architectures used in generative applications use autoregressive decoding. In this scheme, the model is provided a prefix or prompt to use to generate additional text. The model then generates a single token and appends this generated token back onto the prefix. This process is repeated until a maximum sequence length is reached or the model generates a special end token. Based on this approach, traditional autoregressive decoding requires n forward passes of the entire model to generate n tokens. While this framework is feasible when the model is small and a single forward pass is relatively inexpensive, utilize modern LLMs with billions of parameters quickly becomes impractical with limited compute.

Speculative decoding offers an alternative setup to inference where a much smaller "draft model" is used to generate tokens autoregressively [3, 4]. Then the original LLM, or the "target model", verifies the generated tokens in parallel using a single forward pass. The tokens that match the target model's output are accepted and the remaining tokens are rejected. In this way, a single forward pass of the target model can lead to multiple decoded tokens. While this inference framework can theoretically lead to improvements in inference time, the performance of speculative decoding is highly contingent on the choice of draft model. A draft model that is too large would make the draft model token generation step slow and inefficient. A draft model that is too small may be a poor representation of the target model meaning few of the tokens generated by the draft model are accepted by the target model. Speculative decoding literature generally assumes that a draft model with the same vocabulary and training data as the target model already exists. For example, recent literature has used LLaMA-160M as the draft model for Vicuna-7B [5] or OPT-125M as the draft model for OPT-13B and OPT-30B [6]. Our final project instead aims to test methods to construct a draft model for an arbitrary target model. A draft model must accurately mimic the output distribution of the target model on a variety of inputs. To achieve this, we test multiple different model compression techniques and use the compressed model

as the draft model in a speculative decoding inference setup. We make the observation that naively applying lossy compression generally leads to a draft model that does not effectively mimic the behavior of the target model. However, choosing the method that best balances model utility and model size generally leads to improvements in inference efficiency across a wide array of input prompts and maximum sequence lengths.

2 Related Work

LLM inference efficiency has become a cornerstone of ML research [7]. As LLMs have become the predominant method for most natural language tasks, recent literature has focused on determining ways to efficiently serve LLMs. In traditional autoregressive decoding, a prefix sequence of n tokens is passed to the model resulting in the generation of n sets of logits. The next token is sampled from the last logits in this set of n logits. Once the token is sampled, this token is appended onto the prefix to perform the same sequence of steps with a prefix that is now $n + 1$ tokens in length. A consequence of this process is that in order to decode k tokens, the model must perform k forward passes. Since this kind of inference cannot be parallelized efficiently, traditional autoregressive inference tends to be a high memory-bandwidth operation [8]. Speculative decoding has emerged as a technique to optimize LLLM inference [3, 4]. These works suggested using a smaller more efficient draft model to perform autoregressive decoding. The original LLM then acts as a "verifier", performing a single forward pass and simultaneously accepting and rejecting a portion of the draft model's guesses. Early works in speculative decoding were specifically focused on inference speedups and largely ignored choice of draft model. For example, [4] trains a 4 billion parameter draft model from scratch to perform speculative inference for Chinchilla [9]. Similarly, [3] explains that they use existing off-the-shelf smaller transformers as their choice of draft model, such as using T5-Small, which is trained on English-German translation tasks, as the draft model WMT EnDe. Other works on speculative decoding similarly use preexisting draft models trained on the same task for specific target models and instead focus on the method used to perform speculative decoding. For example, [5] recommends retraining the draft model using knowledge distillation based on user queries to better align the output distribution of the draft model with that of the target model. This online speculative decoding approach uses LLaMA-160M as the draft model for Vicuna-7B. The approach outlined in [10] suggests using a tree of speculative tokens as opposed to a single sequence and uses dynamic programming to find the optimal tree structure. This method uses LLaMA-68M and Sheared-LLaMA-1.3B as the draft models for LLaMA-13B and Vicuna-33B. Other approaches aim to construct an effetive draft model for a specific target model. For instance, [11] focuses on benchmarking wider and shallower variants of Sheared-LLaMA-1.3B as potential draft models for LLaMA-7B. The method proposed in [12] focuses on developing LlaMA-2-Chat-Drafter-115M, a draft model for LLaMA-7B. Each of these methods aims to test a distinct approach for speculative decoding with preexisting draft-target model pairs or builds a draft model for a specific instance of target model. Our final project differs from these previous works by examining draft model generation for arbitrary target models. Concretely, the methods for draft model generation described below can be used for any target model that uses autoregressive decoding given that we know the tokenizer used by the target model. Any approach that involves speculative decoding can use these draft model generation techniques so that the method can be tested more robustly with target models that do not already have a draft model with the same vocabulary and training dataset.

3 Methods

3.1 Speculative Decoding

We follow the inference procedure outlined in [4]. Consider an initial prefix of n tokens, $x_1, x_2, \ldots, x_n$. We additionally have a draft model which we refer to as d and a target model which we refer to as t. The process starts by using d to generate k tokens, $x_{n+1}, x_{n+2}, \ldots, x_{n+k}$, autoregressively. From this draft model generation step, we get $k + 1$ sets of logits.

$$d([x_1, x_2, \ldots, x_n]), d([x_1, x_2, \ldots, x_n, x_{n+1}]), \ldots, d([x_1, x_2, \ldots, x_{n+k}]) \tag{1}$$

Each $d([x_1, x_2, \ldots, x_{n+i}])$ is of size $\mathbb{R}^{(n+i)\times v}$ where v is the vocabulary size of both d and t. However, we only need to retain the $(n + i)$th set of logits from each of these representations. The k generated tokens are appended to the prefix of n tokens resulting in a new prefix of $n + k$ tokens $x_1, x_2, \ldots, x_{n+k}$. These $n + k$ tokens are used as input to the target model resulting in $t([x_1, x_2, \ldots, x_{n+k}])$ where $t([x_1, x_2, \ldots, x_{n+k}]) \in \mathbb{R}^{(n+k)\times v}$. We then use the logits $t([x_1, x_2, \ldots, x_{n+k}])$ to verify tokens $x_{n+1}, x_{n+2}, \ldots, x_{n+k}$. In particular, we sequentially check each token x_{n+i}. For each such token, we sample $r \sim U[0, 1]$. Then, we pick the logit corresponding to token x_{n+i} in $d([x_1, x_2, \ldots, x_{n+i-1}])$, which we call l_d, and the logit corresponding to token x_{n+i} in the $(n + i - 1)$st row of $t([x_1, x_2, \ldots, x_{n+k}])$, which we call l_t. To decide whether or not the token x_{n+i} is accepted, we determine if $r < \min(1, \frac{l_d}{l_t})$. If r satisifies this condition, the token is accepted. Otherwise, the token is rejected and a new token is

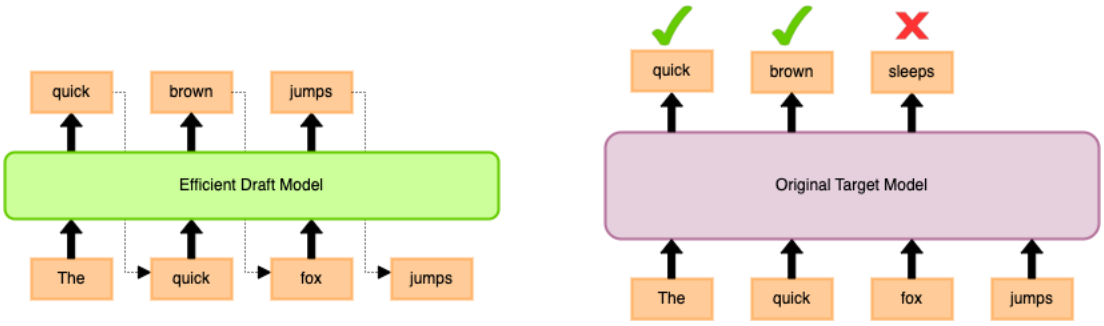


Figure 1: Speculative decoding procedure involves a draft model that performs autoregressive decoding and a target model that verifies the generated tokens.

sampled from the difference of the logits between the target model and the draft model. If all k tokens are accepted, and additional $(k+1)$st token is sampled from the target model logits. This process is repeated until a maximum sequence length is reached or a special end token is generated. There are three notable aspects of this methodology. First, a token is only accepted if it agrees with the target model's output distribution. Second, even if no tokens are accepted, a token is generated. Therefore, in the worst case, speculative decoding ensures that it decodes at least as many tokens as traditional autoregressive decoding. Finally, the target model can verify multiple tokens in parallel using only a single forward pass.

This method of sampling additionally ensures that the output of inference matches the distribution of the original target model despite using the draft model for the majority of inference. This process elucidates the importance of draft model choice. If the draft model does not accurately mimic the output distribution of the target model, few tokens will be accepted by the target model. As a consequence, even though the draft model performs k forward passes to generate k tokens, most of this computation will be wasted. On the other hand, if the draft model is large, performing k forward passes of the draft model will be nearly as expensive as inference using the target model. For this reason, balancing model size and model utility for the draft model is critical in actually speeding up inference using speculative decoding.

3.2 Model Compression

Model compression techniques aim to reduce the size of an ML model by altering the weights or structure of the model [13]. We test various model compression techniques to generate a draft model for an arbitrary LLM. Specifically, we test model pruning, quantization, and layer compression in a speculative decoding inference framework. We explain these different techniques below.

Model Pruning Model pruning involves scoring the weights of the model according to a scoring function and setting the weights with the lowest scores to 0. By reducing the size of the model and eliminating weights that are less necessary to the model performance, we construct a model that is more efficient while retaining the original model's utility. We divide model pruning into two categories: structured pruning and unstructured pruning.

- In structured pruning, the scoring function ranks existing structures within the model architecture (filters, channels, layers, etc.) [14]. Structures are then pruned in their entirety by setting every parameter that exists in that structure to 0. Structured pruning is useful in accelerating training because block level sparsity can speed up matrix multiplication on GPUs. However, the downside to structured pruning is that there is less flexibility in pruning patterns, often resulting in a model that is significantly less accurate than its dense counterpart.
- In unstructured pruning, the scoring function ranks individual weights within the model architecture. Weights with the lowest scores are set to 0 independently of other weights in the model. While unstructured pruning offers more flexibility in sparsity patterns and usually results in more accurate models, they offer few benefits in terms of training efficiency and inference latency. This is because random sparsity patterns require custom hardware to see noticeable speedups for operations like matrix multiplication. For general consumer hardware, there are few efficiency advantages to applying unstructured pruning.

We test both structured and unstructured pruning in our speculative decoding framework. For structured pruning we partition weight matrices into blocks of size $n \times n$ if the weight matrix is 2-dimensional and blocks of size n if the weight matrix is 1-dimensional. Blocks are then pruned by setting every weight in the block to 0. We additionally consider a form of structured pruning where n out of every m contiguous weights is pruned as described in [15]. For our

scoring function we use the $l1$ norm of the weights. This applies to both structured and unstructured pruning. Weights are pruned by creating a binary mask with 1s at the position of the weights or blocks with the highest $l1$ norm and 0 everywhere else. Weight matrices are element-wise multiplied with the binary mask.

Quantization Quantization describes methods that represent the weights in a weight matrix with values at a lower bit precision [16]. By reducing the memory footprint and computational complexity of the model, quantization can significantly accelerate inference times and decrease power consumption. Furthermore, when carefully implemented, quantization can maintain a high level of model accuracy, making it an essential technique for deploying complex models in resource-constrained environments. We implement two forms of post-training quantization, namely GPTQ and NF4. GPTQ uses approximate second-order information to perfom post-training quantization [17]. The GPTQ method involves 3 crucial steps. First, GPTQ uses Cholesky kernels to calculate the inverse Hessian of a weight matrix. Next, the method iterates through blocks in the weight matrix quantizing columns in each block. Finally, the quantization error, which is formulated as $||\mathbf{WX} - \widehat{\mathbf{W}}\mathbf{X}||_2^2$ where $\mathbf{W}$ is the original weight matrix, $\mathbf{X}$ is the input, and $\widehat{\mathbf{W}}$ is the quantized weight matrix, is calculated and used to update the remaining weights in the block. Ultimately, this process iteratively quantizes weights in each layer of the model and then reassignes the remaining weights to reduce the quantized error. 4-bit NormalFloat Quantization (NF4) quantizes values in a $[-1, 1]$ range [18]. NF4 quantization similarly has 3 crucial steps. First, estimates 2^k quantile boundaries where k is the number of bits we use for quantization. These boundaries are estimated as $q_i = \frac{1}{2}(Q_X(\frac{i}{2^k+1}) + Q_X(\frac{i+1}{2^k+1}))$ where Q_X is the quantile function of the standard normal distribution. Second, this k-bit datatype is normalized into the $[-1, 1]$ range. Finally, for an input tensor, which in this case would be the weight matrix for a layer in the model, would also be normalized into the $[-1, 1]$ range using using absolute maximum rescaling and quantized based on our new datatype.

Layer Compression The final method that we implement to perform model compression is layer compression. This method is built around the idea that a Transformer architecture is constructed from a set of identical transformer "blocks". Each of these blocks traditionally is comprised of some form of Multi-Head Attention as well as a feedforward neural network. Crucially, larger transformers are simply built by including more of these blocks in their architecture. The idea behind layer compression is that we can build a smaller transformer by specifying a number of blocks that we hope to keep and then taking the weights from alternating layers of the transformer. For example, in a 12-block transformer architecture, if we hope to construct a smaller transformer with only 3 blocks, we can take the weights from blocks $[0, 6, 12]$ to compress the original model into a 3-block architecture. Extrapolating to an arbitrary n-block architecture, if we aim to build a transformer with k blocks, we can take the layer weights from blocks $[0, \frac{n}{k-1}, 2\frac{n}{k-1}, \ldots]$ to buld this smaller, more efficient transformer.

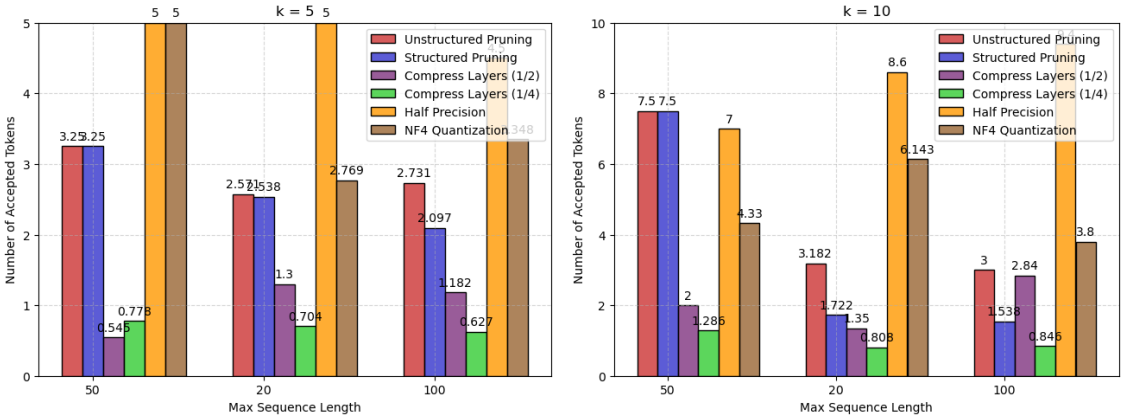
3.3 Benchmark

To test the effectiveness of these methods we use two autoregressive decoding models, namely GPT2 [19] and LLaMA2 [20]. Note, that the baseline performance of these two models is very different with LLaMA2 being a much more expressive model than GPT2. We test the models with the following prompts listed in the below table -

Description	**Sentence Prefix**
General Context	"Today, the weather is"
Specific Knowledge	"The capital of France is"
Creative Continuations	"Once upon a time, in a land far away,"
Technical Descriptions	"In Python, a list can be created by"
Philosophical Queries	"What is the meaning of life"
Logical Reasoning	"If all cats are animals, and all animals have hearts, then"
Conversational Style	"Hey, how was your day today?"
Historical Facts	"During World War II,"
Formal Writing	"The purpose of this document is to"
Humorous Tone	"Why did the chicken cross the road?"

Table 1: Examples of Sentence Prefixes for Testing Autoregressive Decoders

The goal is to demonstrate this method's robustness across various choices of target model as well as various choices of input prompt. We now describe the evaluation of the method across these different configurations.



(a) Draft model generates 5 tokens before verification (b) Draft model generates 10 tokens before verification

Figure 2: Average number of accepted tokens across our various prompts.

4 Results

4.1 Number of Accepted Tokens

We first test our various compression methods by measuring the number of accepted tokens. For both models, we construct a draft model using a specific compression technique. The target model verifies the k tokens generated by the draft model and accepts a subsequence of them. After the maximum sequence length is reached, or a specific end token is returned, we return the average number of accepted tokens out of each set of k tokens generated by the draft model. We find that the technique used in model compression greatly impacts the average number of accepted tokens. For example, half precision and quantized models lead to a higher number of accepted tokens by the target model whereas layer compression significantly reduces the draft model's utility. The performance of model pruning depends heavily on the sparsity ratio used. We find that a sparsity ratio of 0.25 adequately balances model performance with model size. Additionally, unstructured pruning performs better than structured pruning. However, using a smaller block size for structured pruning yields results that are close to unstructured pruning on many benchmarks. We include further tests where we compose compression methods like quantization and model pruning for LLaMA2 in the Appendix.

4.2 Inference Speedups

In addition to the number of accepted tokens, we look at wall-clock improvements for inference. Draft models that more closely mimic the target model tend to be larger leading to slower inference. For example, while half-precision draft models tend to closely match the output distribution of the full-precision target models, they perform inference slower than a layer-compressed draft model with $\frac{1}{4}$ of the original target model's layers. However, because half-precision leads to more accepted tokens, this slower inference is justified by the performance of the draft model. We find that the draft models that perform best are built using half-precision or NF4 quantization. Structured pruning draft models see smaller speedups, but still improve inference time. Unstructured pruning does not speedup the model as inference with this draft model is just as slow as the original target model. Finally, draft models built using layer compression may actually be slower as even though the draft models are more efficient, few tokens are ever accepted by the target model. Ultimately, choosing the method that best balances model size with utility consistently leads to improvements in inference time across various values of sequence length. Results for inference time improvements can be found in the Appendix.

5 Conclusion

Speculative decoding is a method used to improve the inference efficiency of autoregressive decoding. However, many speculative decoding papers assume that a draft model with the same training dataset and vocabulary as the target model already exists. Our final project tests various model compression techniques to generate a draft model for an arbitrary target model. We find that certain compression techniques lead to a high number of accepted tokens as well as improvement in wall-clock inference time. This demonstrates that speculative decoding is a feasible approach for improving LLM inference even without a preexisting draft model. We hope to incoporate more sophisticated methods for compression in future work.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[2] Salman Mohamadi, Ghulam Mujtaba, Ngan Le, Gianfranco Doretto, and Donald A. Adjeroh. Chatgpt in the age of generative ai and large language models: A concise survey, 2023.

[3] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023.

[4] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023.

[5] Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Ion Stoica, Zhijie Deng, Alvin Cheung, and Hao Zhang. Online speculative decoding, 2023.

[6] 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. 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*, ASPLOS '24. ACM, April 2024.

[7] Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, Shengen Yan, Guohao Dai, Xiao-Ping Zhang, Yuhan Dong, and Yu Wang. A survey on efficient inference for large language models, 2024.

[8] Noam Shazeer. Fast transformer decoding: One write-head is all you need, 2019.

[9] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022.

[10] Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable, robust, and hardware-aware speculative decoding, 2024.

[11] Minghao Yan, Saurabh Agarwal, and Shivaram Venkataraman. Decoding speculative decoding, 2024.

[12] Raghavv Goel, Mukul Gagrani, Wonseok Jeon, Junyoung Park, Mingu Lee, and Christopher Lott. Direct alignment of draft model for speculative decoding with chat-fine-tuned llms, 2024.

[13] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression, 2017.

[14] Ziheng Wang, Jeremy Wohlwend, and Tao Lei. Structured pruning of large language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2020.

[15] Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.

[16] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference, 2017.

[17] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023.

[18] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms, 2023.

[19] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[20] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan

Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

6 Appendix

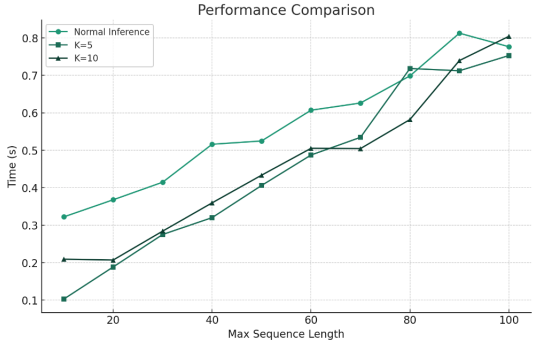


Figure 3: Improvements in inference time when choosing the fastest applicable method.

Table 2: Evaluation of speculative decoding for different generation lengths and precision (draft/target) models.

	Len=120		Len=240			
	8/32	16/32	8/16	8/32	16/32	8/16
Prompt-A	1.0	1.0	0.96	0.97	1.0	0.97

Table 3: Evaluation of speculative decoding for different pruning methods (unstructured pruning and 4:8 structured pruning) with generation lengths of 240 tokens, and precision (draft/target) models. Combining extreme sparsity and quantization leads to much lossier compression where the target model on average does not accept many tokens.

	0.5-Unst			0.3-Unst			4:8		
	8/32	16/32	8/16	8/32	16/32	8/16	8/32	16/32	8/16
Prompt-A	0.0	0.4	0	0.13	0.73	0.02	0.37	0.33	0.87

Table 4: Quantization Methods for LLaMA (k=5)

Max Sequence Length	**GPTQ**	**NF4**
50	5	2.23
20	5	2.76
100	5	2.39

Table 5: Quantization Methods for LLaMA (k=10)

Max Sequence Length	**GPTQ**	**NF4**
50	10	3.16
20	10	3.8
100	10	2.7