The N-Grammys: Accelerating Autoregressive Inference with Learning-Free Batched Speculation

Lawrence Stewart[*] ENS Paris
Matthew Trager AWS AI Labs
Sujan Kumar Gonugondla[†] AWS NGDE Science
Stefano Soatto AWS AI Labs

Abstract

Speculative decoding aims to speed up autoregressive generation of a language model by verifying in parallel the tokens generated by a smaller draft model. In this work, we explore the effectiveness of learning-free, negligible-cost draft strategies, namely N-grams obtained from the model weights and the context. While the predicted next token of the base model is rarely the top prediction of these simple strategies, we observe that it is often within their top-k predictions for small k. Based on this, we show that combinations of simple strategies can achieve significant inference speedups over different tasks. The overall performance is comparable to more complex methods, yet does not require expensive preprocessing or modification of the base model, and allows for seamless 'plug-and-play' integration into pipelines.

1 Introduction

Large Language Models (LLMs) have had a significant impact across various scientific and industrial domains. However, their autoregressive decoding process, which generates one new token per model call, is computationally expensive. This issue is particularly challenging for larger models, which typically exhibit superior performance compared to smaller ones [Brown et al., 2020, Anil et al., 2023, Achiam et al., 2023].

To improve computational efficiency and inference latency, many works have proposed methods for reducing the cost of a single model call. Some examples include quantization methods Yao et al. [2023], early-exiting strategies Xin et al. [2020], flash-attention [Dao et al., 2022, Dao, 2023], and multi-token prediction [Gloeckle et al., 2024].

Another line of work has considered variants of autoregressive decoding aimed at better leveraging the parallel processing capabilities of GPU/TPU hardware accelerators. In particular, *speculative decoding* methods [Stern et al., 2018, Leviathan et al., 2023, Chen et al., 2023a, Xia et al., 2023], sometimes also called *guess-and-verify* methods, use a smaller "draft model" to generate proposals for multiple future tokens. They then "validate" all of these tokens in parallel with a single call to the original model and ensuring that the original model would have predicted the same tokens. This approach is similar to speculative execution [Hennessy and Patterson, 2011], in which a processor executes instructions in parallel to verifying if they are needed, trading resources for concurrency, as in just-in-time XLA compilation in JAX, Pytorch, and Tensorflow.

The effectiveness of speculative decoding is determined by i) the acceptance rate of the draft speculations, ii) the discrepancy in call-time between the original model and the draft model, and iii) the extra cost required for parallel verification by the main model (although this is often assumed to be negligible when using hardware accelerators).

[*]Work done during an internship at AWS AI Labs.
[†]Currently at Meta AI.

4th NeurIPS Efficient Natural Language and Speech Processing Workshop (ENLSP-IV 2024).

Choosing a draft model that is compatible with the base model and within the available compute budget can be challenging. To address this issue, several approaches have been developed that involve augmenting the base model and performing supervised fine-tuning (SFT), with the goal of ensuring that the draft model and the base model utilize the same feature representations [Cai et al., 2024, Li et al., 2024, Bhendawade et al., 2024]. Although these strategies often achieve high acceptance rates, they come with the drawback of necessitating SFT for each individual model, which can be resource-intensive.

To overcome this, researchers have also considered *negligible cost draft models*, remarking that if the cost of the draft model is close to zero then even a low acceptance rate can yield wall-time speedups. This was first proposed explicitly in Leviathan et al. [2023], where the authors experimented with unigram and bigram draft models, both trained on external data, as well as implicitly by Santilli et al. [2023], who used a cost-free draft model, corresponding to the base model's greedy predictions at a previous time step.

In this paper, we aim to explore the full potential of negligible-cost draft methods for accelerating autoregressive decoding. In particular, we argue that even simple strategies — based on N-grams derived from the model and the context — can be very effective when combined in a batch to explore the space of possible future trajectories in parallel. Our proposed methods have the following desirable features: **(P1)** they do not require training a draft model or finetuning the base model, **(P2)** they make use of no external data or external draft model, and most importantly **(P3)** they can easily be integrated with any existing pipeline as an *out-of-the-box* approach, and moreover they can be combined with other acceleration techniques like the ones mentioned above.

We emphasize, the goal of this paper is not to achieve state-of-the-art inference speed-ups but rather explore the strengths and weaknesses of simple methods that satisfy the desirable properties **(P1)**, **(P2)**, **(P3)**. Our experiments across different datasets (MTBench Zheng et al. [2023], HumanEval Chen et al. [2021], GSM8K Cobbe et al. [2021]) and models (Mistral7B Jiang et al. [2023], Phi-3 Abdin et al. [2024], Vicuna13B Zheng et al. [2023]) show that such *out-of-the-box* strategies are surprisingly effective.

Main contributions.

1. We first revisit the critical assumption made by guess-and-verify methods, namely memory-bound hardware parallelism, highlighting situations where this assumption may fail.
2. We present a class of learning-free strategies for generating batches of speculative drafts with negligible computational cost. These strategies are model-independent and can be implemented with minimal wrapper-code, enabling easy integration into existing systems.
3. We conduct an in-depth analysis and evaluation of our methods, demonstrating that combining simple strategies can lead to substantial speedups across a diverse set of tasks.

2 Further related work

External draft model. The concept of guess-and-verify using an *external draft model* was explicitly proposed in several concurrent works [Leviathan et al., 2023, Chen et al., 2023a, Xia et al., 2023], with Leviathan et al. [2023] notably exploring negligible cost models and suggesting that fitting N-grams to the context could potentially be a promising line of future work. Recently, follow-up works have investigated using a collection of varied size draft models [Chen et al., 2023b], tree-based guess-and-verify methods [Miao et al., 2023], retrieving speculations from external data sources [He et al., 2023], and using a collection online-buffers for aligning the draft and base model via training [Liu et al., 2023]. Contrary to these works, we aim to explore strategies that do not require an external draft model.

Learning by adapting the base model. In order to align the predictions of the draft model with those of the base model, several works have proposed grafting a draft model on top of the base model, so that both share the same features. Cai et al. [2024] propose adding K heads to a model in order to predict K tokens into the future, together with a tree-based attention mechanism. The authors explore i) fine-tuning only the heads with the base-model frozen ii) fine-tuning the base LLM with the heads. Building on this idea, Li et al. [2024] proposes training an auto-regressive decoder from the penultimate layer, showing that this approach can obtain very high acceptance rates.

Learning-free methods. Santilli et al. [2023] proposed initializing a random speculation, and at subsequent decoding steps using the model predictions from the previous step as speculations, in order to improve upon greedy decoding. This process resembles the Jacobi and Gauss-Seidel iterative methods (and is hence called Jacobi Decoding), and is implementable just a few lines of code. Look-ahead decoding Fu et al. [2024] further improves upon the acceptance rate of Jacobi decoding, by using a custom-attention mask to generate an N-gram speculation cache as well as verifying matching speculations in parallel.

Glossary

Symbol	Usage
$\mathcal{X}$	the set of tokens that constitute the vocabulary of an LLM.
k	number of batched speculations, taken from the top-k of probability over tokens.
w	number of tokens speculated into the future.
q	number of query tokens to match with context when looking for an N-gram.
ℓ	length of context at a decoding step, assumed to be key-valued cached (KV-cached).

Limitations. For simplicity and ease of integration, our method incurs extra computation cost through batching (which could be addressed in follow-up works by incorporating methods such as bifurcated attention Athiwaratkun et al. [2024]). Further exploration is needed for non-greedy sampling methods such as those discussed in [Leviathan et al., 2023], which are commonly deployed. Finally, we have limited our experiments to decoder-only transformer models, and it remains to be tested with other architectures such as state-spaced models [Gu and Dao, 2023].

3 Assumptions on parallelism for verification

We briefly revisit and clarify the key assumption in the *guess-and-verify* literature [Leviathan et al., 2023, Chen et al., 2023a, Xia et al., 2023, Santilli et al., 2023] that parallel verification of speculated tokens by the base model is memory-bound when using hardware accelerators like GPUs/TPUs.

Hardware accelerators like GPUs and TPUs divide matrix multiplications (matmuls) into tiles, each assigned to independent computation threads. These operations can be parallelized if their operations-to-bytes ratio (OTB) is below the hardware's threshold, allowing all threads to be assigned to multiprocessors and executed concurrently. If the OTB ratio is above the threshold, the operation becomes compute-bound (or math-bound), as the number of tiles exceeds the number of multiprocessors, requiring the total number of tiles to be quantized to fit the available hardware resources.

For a fixed model and hardware, let ℓ denote the length of the given context at any decoding step, with all context tokens (except for the final token), assumed to have KV-cache stored in memory. Let $(k, w+1)$ denote the dimensions of the input batch, where $k \geq 1$ denotes the batch size and $w \geq 0$ denotes the number of tokens speculated into the future. With this terminology, the *guess-and-verify* assumption above can be rephrased as the following: *'for a fixed model using KV-caching and a fixed hardware accelerator, and for a given context length $\ell \geq 1$, the time required to perform a model call on an input block of size $(k, w+1)$ is approximately the same as the time for a model call on an input block of size $(k, 1)$.*

When does this assumption hold? For each element in a batch of size k, the attention mechanism requires multiplying $(w+1)$ queries by $(\ell + w)$ keys, resulting in $\mathcal{O}\left(kw(w+\ell)\right)$ complexity. So for a fixed model, accelerator and (ℓ, k, w) values, the assumption holds if and only if all matmuls in the forward pass of the model have an OTB ratio less than the accelerator's threshold. In practice, this does not always hold. Figure 1 depicts the phase-transition from memory-bound to compute bound, with varied (ℓ, k, w), for Mistral 7B on a NVIDIA A100 40GB GPU. One does not see a smooth scaling of $\mathcal{O}\left(kw(w+\ell)\right)$ in the phase transition, due to the quantization to multiprocessors, resulting in jumps known as *wave quantization*.

In the special case of neglible cost draft models, for which the time to generate speculations is assumed to be near zero, there will be a clear trade-off between the speed-up gained from accepting extra tokens (by increasing k and w) vs the potential slow-down that could be faced when entering a compute-bound setting, where the guess and verify assumption is broken.

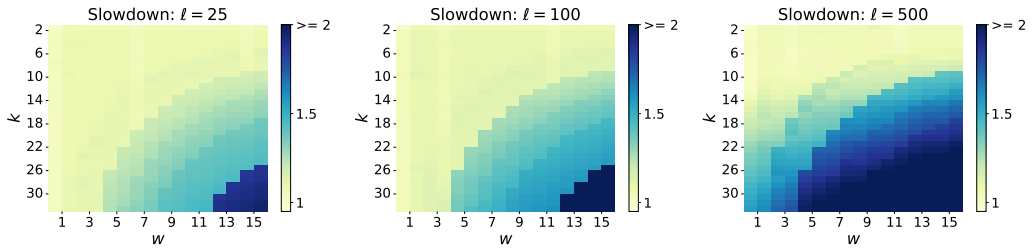

Figure 1: **Memory-bound to compute-bound transition:** The heatmaps depict the slowdown of a model call, for varied batch size $k \in \{1, \ldots, 32\}$ and speculation length $w \in \{0, \ldots, 15\}$. The slow-downs are relative to that of standard greedy decoding with no speculation *i.e.* $(k, w) = (1, 0)$. The leftmost plot corresponds to a context-length of $\ell = 25$, the middle to $\ell = 100$ and the rightmost plot to $\ell = 500$. The model used was Mistral 7B at standard bfloat-16 precision, with a single NVIDIA A100 GPU with 40GB of memory. Each square in the heat-maps corresponds to the average slow-down over five model calls.

4 Learning-free drafting

The use of N-grams to model language dates back at least to Markov [1913], who published a paper in which he used conditional probabilities of constants and vowels (computed by hand) to compare the poem *Eugene Onyegin* by Pushkin to other texts, showing how unigram and bigram probabilities could mathematically capture an author's style. Many sequences of tokens in natural language / computer code exhibit low entropy, making even simple N-grams effective at predicting them, as demonstrated in Shannon [1951]. In this section, we explore ways to extract N-grams directly from a large language model and a context, and use these for speculation. These methods are learning-free, as they require no training **(P1)**, nor external data **(P2)**, in contrast to the N-gram models used in Leviathan et al. [2023], which are obtained from external data sources. All the methods discussed in this section can be implemented with minimal wrapper code (detailed in Appendix B), allowing for them to be added to existing pipelines with minimal friction **(P3)**.

4.1 Model-derived N-grams

Let M denote a language model with vocabulary $\mathcal{X}$. Let $V \in \mathbb{R}^{\mathcal{X} \times d}$ and $U \in \mathbb{R}^{d \times \mathcal{X}}$ denote the model's input and output embedding layers, with respective row / column word embeddings $\{v_i\}_{i \in |\mathcal{X}|}$ and $\{u_i\}_{i \in |\mathcal{X}|}$. For a given context c (i.e. a sequence of tokens from $\mathcal{X}$), we denote the next token distribution according to M as $p_M(\cdot|c)$, where $\sum_{x \in \mathcal{X}} p_M(x|c) = 1$.

Unigram. Consider the function $d(x) = \|u_x - \bar{u}\|_V$, where $\bar{u} \in \mathbb{R}^d$ is the mean token output embedding, and $\|\cdot\|_V$ is the distance induced by the covariance matrix of the input embeddings V, that is, the inner product $\langle u_1, u_2 \rangle_V = u_1^T V^T V u_2 = \sum_{x \in \mathcal{X}} (u_1^T v_x)(u_2^T v_x)$. This product is more natural than the standard one in $\mathbb{R}^d$ as two tokens will be close when they lead to similar distributions for the following token for the model, as captured by the vectors $(u_i^T v_x \colon x \in \mathcal{X}) \in \mathbb{R}^{|\mathcal{X}|}$. We can hence define a unigram distribution over tokens using the input and output embeddings as $p(x) \propto e^{-d(x)}$.

Bigram. We can easily obtain a bigram model from a language model M by calculating $p_M(\cdot \,|\, x)$ for all tokens $x \in \mathcal{X}$. For typical models, this can be a calculated once for every $x \in \mathcal{X}$ and stored for quick use later. For example, generating such a bigram model takes ≤ 1 minute for Mistral 7B on a single A100 GPU, and is a one-off cost. While this simple bi-gram lacks context for tokens prior to x, it can still be effective, particularly in cases where the preceding context is not essential for making accurate predictions.

Batched drafts. When performing autoregressive decoding, the greedy next token prediction (NTP) of the base model will seldom match with that predicted by the above unigram and bigram models (derived from the base model). However, we remark that the base model NTP appears often amongst the top-k predictions of the N-grams, even for small k. Consequentially, we propose obtaining

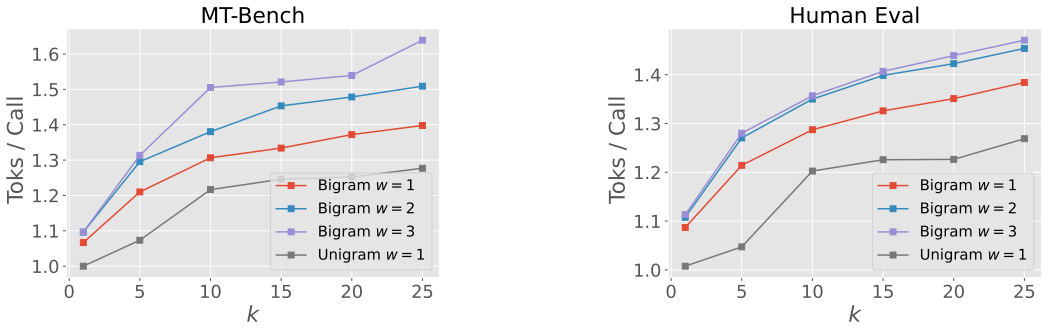

Figure 2: Tokens per call as a function of k, the top-k speculations of the model derived unigram / bigram. In addition, the plot depicts the extended bigram (described below) plotted for $w = 2$ and $w = 3$, showing gains comparing $w = 1$ to $w = 2$, but diminishing gains going to $w = 3$. The results were obtained on the first 50 examples of MT-bench and Human Eval, using a 7B model (Mistral Instruct) [Jiang et al., 2023].

speculations from the top-k of a N-gram model, i.e., $s : \mathcal{X}^q \to \mathcal{X}^{k\times 1}$, where $q \geq 0$ denotes the number of last context tokens to use to produce the speculation (i.e. $q = 0$ for the unigram and $q = 1$ for the bigram), and the speculation s returns the top-k *next-word* token predictions according to the N-gram.

Speculative decoding using our model derived unigram / bigram can be easily implemented in the following manner: i) repeat the context[3] to form a batch of k identical rows; ii) append a column corresponding to the top-k speculations to the end of the batch; iii) call the model on the batch to verify all speculations (rows) in a single forward pass.

While adding redundant computation (regarding repeat flops for the context), this implementation is *extremely simple* to integrate into existing code, and in addition, is fully-compatible with popular inference methods, *e.g.* , flash-attention [Dao et al., 2022, Dao, 2023] and paged-attention / vLLM [Kwon et al., 2023], which is generally not the case for methods that require custom attention masking such as tree / lookahead attention masking [Fu et al., 2024, Cai et al., 2024].

Extensions. The model-derived bigram and unigram allow for speculating $w = 1$ token into the future. In addition, by repeatedly applying either the model bigram (or, alternatively greedily, decoding with the model from the bigram), we can easily extend the model-derived N-grams to speculate $w > 1$ tokens into the future $\tilde{s} : \mathcal{X}^q \to \mathcal{X}^{k\times w}$. Just like the model bigram, this extension can be generated quickly one time and stored as a $O(1)$ lookup table.

Top-k speculation. Figure 2 provides evidence motivating our approach, depicting how speculating with the top-k tokens increases the number of tokens per call for a 7B model on the first 50 examples of MT-Bench and Human Eval, for both the unigram and bigram models, obtained directly from the Transformer. We remark that with $w = 2$, speculating with the top-25 of the model bigram gives a $\approx 50\%$ increase in tokens per call.

4.2 Context-derived N-grams

Another natural idea to obtain N-grams for speculations is to look within the context provided to a model; indeed this was suggested by Leviathan et al. [2023] as a potential avenue for future work. We propose looking for all previous occurrences of the last $q \geq 1$ tokens of the context, and speculating with the $w \geq 1$ tokens that follow a match. To define a discrete probability distribution, we can assign each match a count *i.e.* how many times it occurred in the context, with ties being decided by which match occurred later in the context (hence prioritizing more recent matches). For more details on the context N-gram please refer to the attached code in Appendix B.2.

[3]If the context has a key-value cache, one can utilize a method to obtain a batched tensor view without consuming additional memory, e.g., *torch.extend*.

4.3 Mixed strategies

For a chosen batch size $k > 1$, one has the flexibility of using any combination of "strategies" *i.e.* model / context derived N-grams, to populate the batch of speculations. This allows for the exploration of diverse combinations of speculation methods. In this work we consider the a straightforward way to mix strategies: first, we populate the k drafts by using as many speculations from the context derived N-gram model as possible, depending on how many matches are obtained (possibly zero); then we use a 'extended model bigram' to fill in the remaining speculations. This means that the number of speculations allocated to each strategy (context/model bigram) is variable depending on the context at each step of decoding, which we ablate in Section 5.2.

5 Experiments

Datasets and models. We assess our proposed mixed strategies described in Section 4.3 using the same experimental setup and datasets as Cai et al. [2024], Li et al. [2024]: MTBench Zheng et al. [2023] (a multi-turn question benchmark with many unique tokens), HumanEval Chen et al. [2021] (a coding benchmark), GSM8K Cobbe et al. [2021]) (mathematical reasoning problems). We experimented with three different instruction-tuned models of various sizes: Phi-3 Abdin et al. [2024], Mistral7B Jiang et al. [2023] and Vicuna13B Zheng et al. [2023]. All models are freely available from the Hugging-Face transformers library, with reference url links detailed in the Appendix C. All experiments were run on a single Nvidia A100 GPU with 40GB of memory at bfloat-16 precision. We report two metrics of interest:

1. *tokens per call*: measures how many tokens are produced in a single model call on average, i.e., the acceptance rate. This would be the observed speed-up if one had both true parallelism and zero cost speculation.
2. *wall-time speed-up*: this is the physical observed speed-up on the hardware. To obtain accurate timings, we used the CUDA Runtime API and ran all experiments three times, reporting the mean and standard deviation.

Mixed strategies. We consider mixed strategies (as detailed in Section 4.2), defined by values $k \in \{1, 5, 10, 20, 25\}$ and $w \in \{2, 4, \ldots, 14\}$, with query length $q = 1$[4] when deriving speculations from the context. The resulting (k, w) grid totals 35 different strategies to assess, whose performance will be dictated by trade-offs between i) token per call acceptance and ii) potential compute-bound slowdowns.

For comparative purposes, we include the results reported by lookahead decoding [Fu et al., 2024], an effective *learning-free* guess-and-verify method, using custom-attention masks to grow an N-gram cache in parallel to verifying speculations. Contrary to look-ahead decoding, our method does not require custom attention masks, due to the naive batching (P3), and is hence fully compatible with methods such as flash-attention [Dao et al., 2022, Dao, 2023], and is comparatively simpler to integrate / implement.

We also include results reported by REST (Retrieval-Based Speculative Decoding) [He et al., 2023], a recent approach which also utilizes negligible draft models. REST requires pre-processed databases to retrieve tree-based speculations. In contrast, our method requires no external data, using only the context and model derived N-grams. We note that it is important to exercise caution when forming exact comparisons between methods and models, since additional factors such as hardware[5], tokenizer and instruction formatting will impact the observed speedups for all reported methods.

5.1 Results

For each dataset and model, we report the strategy that led to the largest wall-time speedup, which we denote (k^*, w^*). As a reference, we also reported $(k, w) = (10, 10)$, to compare how a square input

[4]We experimented with longer query length $q = 2$ and $q = 3$, but observed a degradation in both speed-up and tokens per call across all data sets and models.

[5]Lookahead used a GPU with a higher operations-to-bytes (OTB) ratio than our experiments, whilst the experiments of RAST were conducted on a GPU with a lower OTB ratio.

Model Size	Strategy	MT Bench		Human eval		GSM8K	
		tok/call	speedup	tok/call	speedup	tok/call	speedup
3b	Ours $(10, 10)$	2.17	$2.01^{\pm 0.02}$	2.28	$2.11^{\pm 0.01}$	2.38	$2.30^{\pm 0.02}$
	Ours (k^*, w^*)	2.36	$2.18^{\pm 0.02}$	2.51	$2.34^{\pm 0.02}$	2.41	$2.51^{\pm 0.01}$
7b	Lookahead	–	1.65	–	2.25	–	1.89
	REST	–	1.69	–	2.12	–	–
	Ours $(10, 10)$	2.13	$1.91^{\pm 0.01}$	2.22	$2.04^{\pm 0.01}$	2.16	$2.03^{\pm 0.03}$
	Ours (k^*, w^*)	2.13	$1.91^{\pm 0.01}$	2.19	$2.05^{\pm 0.01}$	2.16	$2.03^{\pm 0.03}$
13b	Lookahead	–	1.51	–	2.26	–	1.72
	REST	–	1.77	–	2.17	–	–
	Ours $(10, 10)$	2.78	$2.31^{\pm 0.01}$	2.89	$2.50^{\pm 0.01}$	2.56	$2.21^{\pm 0.01}$
	Ours (k^*, w^*)	2.68	$2.45^{\pm 0.02}$	2.91	$2.77^{\pm 0.02}$	2.46	$2.32^{\pm 0.01}$

Table 1

block, representative of a default non-optimized choice in our sweep fared compared to (k^*, w^*). Table 1 shows the mean and standard deviation across the three runs, for all models and datasets.

For Mistral7B, the average wall-time speedups for the complete grid of strategies is depicted in Figure 3. The grids all show a clear tradeoff between tokens-per-call (by increasing either k and/or w) and compute-bound slowdowns, with the pattern of Figure 3 shared across the three different tasks suggesting a consistent relationship between (k, w) and speed-up. For reference, the corresponding tokens per call are reported in Appendix A.1. The equivalent plots for both Phi3B and Vicuna13B can be found in Appendix A. For Phi3B, we notably observed the model never reached an OTP ratio that was large enough to incur slowdowns which would outweigh increases in tokens per call. For this reason the optimal values were trivially those of maximum value, *i.e.* $(k^*, w^*) = (25, 14)$; the true optimal speed-up using our batched approach would hence occur at a larger (k, w).

Overall, our methods consistently achieve more than 2x speedup across models and tasks (except for the 7B model on MT bench, which attained a 1.91 times speedup). While the optimal (k^*, w^*) varied between models and datasets, it can be seen that the representative default $(10, 10)$ achieved good performance on all of the settings.

5.2 Ablation on strategies

In order to understand the role that both the model- and context-derived N-grams play in the observed speed-ups, we ablate the Mistral7B experiment for $(k, w) = (10, 10)$ across the three data sets. We explore i) the number of speculated tokens accepted by both the model and context derived N-grams ii) the rank of accepted speculations amongst the top-10 speculations iii) the amount of drafts *i.e.* rows in the batch, that each of the strategies used. The results are depicted in Figure 4.

Our ablations shine light on the strengths and weaknesses of both the model bigram and context derived draft strategies. The model-bigram is robust across all tasks, with an additional 1-2 future tokens frequently found within its top-10 predictions, with the ranking distribution being notably heavy-tailed (relative to that of the context-derived N-gram). The model's bigram weakness lies in its

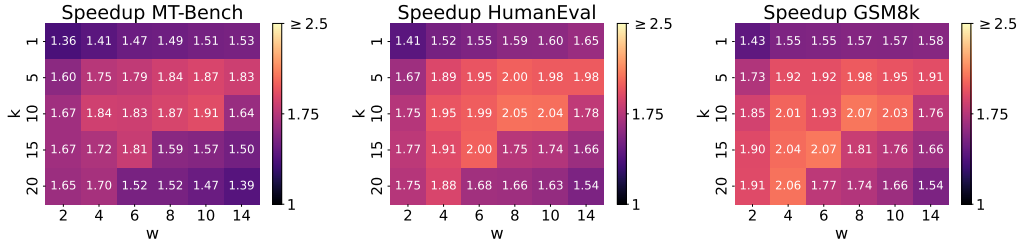

Figure 3: Average wall-time speedup across datasets for Mistral7B instruct for varied (k, w).

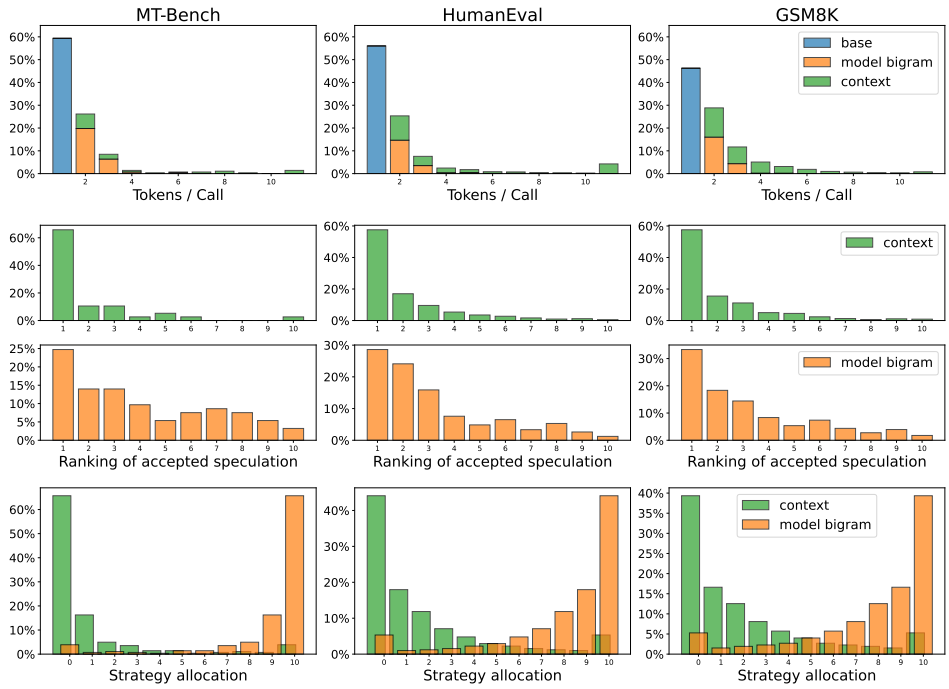

Figure 4: Ablations: **Top**: distribution of acceptance length for mixed strategies. **Middle**: distribution of ranking of accepted speculations amongst the top-k. **Bottom:** allocation distribution of strategies *i.e.* number of speculations for each strategy.

ineffectiveness for larger values of w, which is expected since it only considers the last token of the context rather than encompassing all prior context, making it insufficient for longer speculations.

On the other hand, we see that the context derived N-gram compliments the bigram's weakness as it can successfully speculate further into the future, with speculations of length $w = 10$ accepted on all tasks. However, its performance is notably less robust across tasks. For example, GSM8K exhibits a wider distribution of accepted lengths due to the varied sizes of calculations in the math word-problems, while HumanEval more frequently accepts $w = 10$ length speculations due to the coding nature of the task. Furthermore, it exhibits more pronounced diminishing returns from batching (compared to the bigram), which is a weakness given that it is often allocated the entire batch for speculations (see the bottom row of Figure 4). This suggests that further research into enhancing strategy allocation could indeed yield further additional gains.

6 Conclusions

We introduced a set of learning-free strategies for generating batches of speculative drafts, extracted from both the base model and context. Our approach is conceptually simple and is fully compatible with other optimization techniques (e.g. quantization, early exiting, flash attention, etc.). Experimentally, we observed that our proposed strategies led to significant speedups in auto-regressive inference, while requiring minimal implementation overhead and being easily integrable. Our analysis demonstrates that simple strategy combinations can substantially enhance performance across a range of different tasks and model sizes, with our ablations shining light on the strengths and weaknesses of the proposed strategies.

References

Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

Ben Athiwaratkun, Sujan Kumar Gonugondla, Sanjay Krishna Gouda, Haifeng Qian, Hantian Ding, Qing Sun, Jun Wang, Jiacheng Guo, Liangfu Chen, Parminder Bhatia, Ramesh Nallapati, Sudipta Sengupta, and Bing Xiang. Bifurcated Attention for Single-Context Large-Batch Sampling, March 2024. URL `http://arxiv.org/abs/2403.08845`. arXiv:2403.08845 [cs].

Nikhil Bhendawade, Irina Belousova, Qichen Fu, Henry Mason, Mohammad Rastegari, and Mahyar Najibi. Speculative streaming: Fast llm inference without auxiliary models. *arXiv preprint arXiv:2402.11131*, 2024.

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.

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

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating Large Language Models Trained on Code, July 2021. URL `http://arxiv.org/abs/2107.03374`. arXiv:2107.03374 [cs].

Ziyi Chen, Xiaocong Yang, Jiacheng Lin, Chenkai Sun, Jie Huang, and Kevin Chen-Chuan Chang. Cascade speculative drafting for even faster llm inference. *arXiv preprint arXiv:2312.11462*, 2023b.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training Verifiers to Solve Math Word Problems, November 2021. URL `http://arxiv.org/abs/2110.14168`. arXiv:2110.14168 [cs].

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of llm inference using lookahead decoding. *arXiv preprint arXiv:2402.02057*, 2024.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737*, 2024.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

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

John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7B, October 2023. URL `http://arxiv.org/abs/2310.06825`. arXiv:2310.06825 [cs].

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. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024.

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

Andrey A Markov. An example of statistical investigation of the text eugene onegin concerning the connection of samples in chains. *Science in Context*, 19(04):591–600, 1913.

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*, 1(2):4, 2023.

Andrea Santilli, Silvio Severino, Emilian Postolache, Valentino Maiorca, Michele Mancusi, Riccardo Marin, and Emanuele Rodolà. Accelerating transformer inference for translation via parallel decoding. *arXiv preprint arXiv:2305.10427*, 2023.

Claude E Shannon. Prediction and entropy of printed english. *Bell system technical journal*, 30(1): 50–64, 1951.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models, 2018. URL `https://arxiv.org/abs/1811.03115`.

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

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy J. Lin. Deebert: Dynamic early exiting for accelerating bert inference. In *Annual Meeting of the Association for Computational Linguistics*, 2020. URL `https://api.semanticscholar.org/CorpusID:216552850`.

Zhewei Yao, Cheng Li, Xiaoxia Wu, Stephen Youn, and Yuxiong He. A comprehensive study on post-training quantization for large language models. *ArXiv*, abs/2303.08302, 2023. URL `https://api.semanticscholar.org/CorpusID:257532318`.

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, December 2023. URL http://arxiv.org/abs/2306.05685. arXiv:2306.05685 [cs].

A Additional results

A.1 Mistral 7B

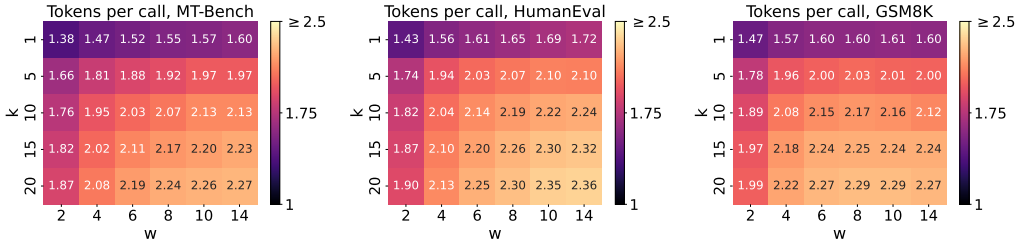

Figure 5: Tokens per call across datasets for Mistral7B instruct for varied (k, w).

A.2 Phi3B

We remark that Phi3B never was compute-bound, so maximum speed-up from mixing strategies was not attained.

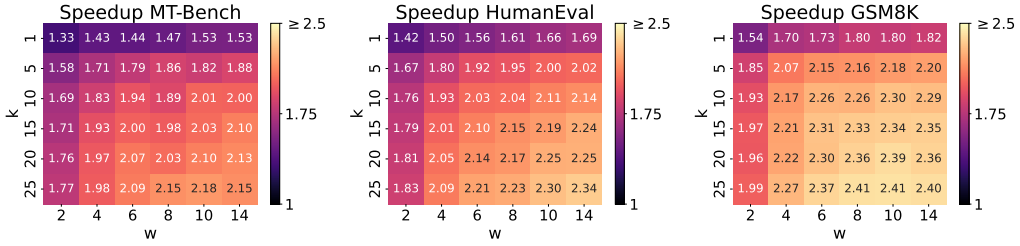

Figure 6: Average wall-time speedup across datasets for Phi3B-instruct for varied (k, w).

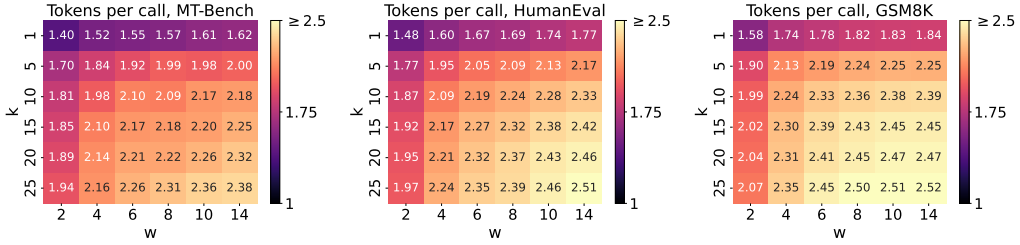

Figure 7: Tokens per call across datasets for Phi3B-instruct for varied (k, w).

A.3 Vicuna 13B

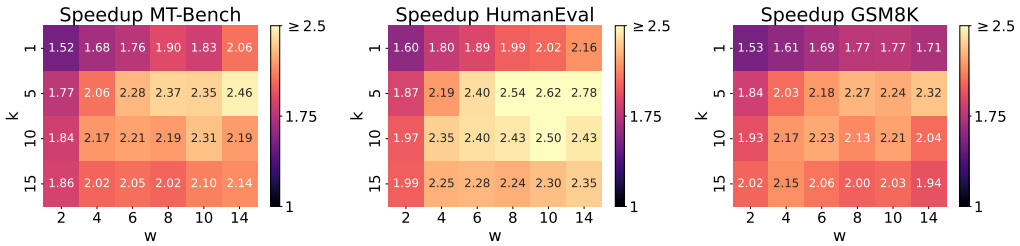

Figure 8: Average wall-time speedup across datasets for Vicuna13B for varied (k, w).

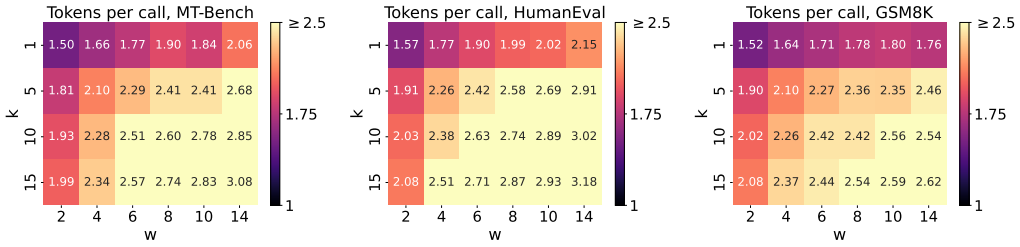

Figure 9: Tokens per call across datasets for Vicuna13B for varied (k, w).

B Learning-Free N-Grams

B.1 Model Based N-Grams

```
def unigram(model):
    '''
    Obtain a Unigram topk from a transformer's weights
    '''
    Wenc = model.get_input_embeddings().weight.detach()
    covV = Wenc.T @ Wenc / Wenc.shape[0]
    Wdec = model.lm_head.weight.detach()

    # look at the distance from the mean embedding in the decoder space.
    mu = Wdec.mean(dim=0, keepdim=True)
    dists = mu @ covV @ Wdec.T
    dists = dists.squeeze()
    ranks = torch.topk(-dists, k=dists.size(0)).indices

def bigram(model):
    '''
    Pseudo code: bigram topk from a transformer:
    '''
    V = model.config.vocab_size

    lookup = torch.empty(V, V)

    # this can be done in batches
    for x in range(len(V)):
        lookup = model(x).logits
    return lookup
```

B.2 Context Based N-Grams

```
@torch.inference_mode
def context_ngram_matcher(context, query, Ndraft=1, Npad=1):
    '''
    matches query of any length Q to grams of size Q + Npad

    query : tensor (query to match with)
    Ndraft : int (number of drafts)
    Npad : int (number of tokens to speculate with)
    '''
    # obtain length of query
    Q = query.size(-1)
    # use unfold to obtain all N grams
    grams = context.flatten().unfold(0, Q + Npad, 1)
```

```
# extract mask of matching ngrams
mask = torch.all(grams[:, :Q] == query, dim=-1)
if torch.any(mask):
    matching_grams = grams[mask]
    # obtain counts of all ngrams
    matches, counts = torch.unique(matching_grams, dim=0, return_counts=True)
    Nfound = counts.size(0)
    Ntake = min(Ndraft, Nfound)
    # take up to top Ndraft occuring Ngrams
    most_freq_ids = counts.topk(Ntake).indices
    return matches[most_freq_ids]
else:
    return None
```

C Models

- Phi3B : `https://huggingface.co/microsoft/Phi-3-mini-4k-instruct` (MIT License)
- Mistral 7B : `https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2` (Apache 2.0 License)
- Vicuna 13B : `https://huggingface.co/lmsys/vicuna-13b-v1.3` (Non-commercial license)

D Key-Value Cache

We use a static key-value cache based upon the implementation from Cai et al. [2024], Li et al. [2024]. However, we add minimal modifications to i) allow for batching ii) over-write all rows to be that of the maximum length accepted speculation iii) initialize from a $k = 1$ cache (since the context is repeated), via a broadcasting.