PaSS: Parallel Speculative Sampling

Giovanni Monea[*]
EPFL
giovanni.monea@epfl.ch

Armand Joulin
Apple
ajoulin@apple.com

Edouard Grave
Apple
egrave@apple.com

Abstract

Scaling the size of language models to tens of billions of parameters has led to impressive performance on a wide range of tasks. At generation, these models are used auto-regressively, requiring a forward pass for each generated token, and thus reading the full set of parameters from memory. This memory access forms the primary bottleneck for generation and it worsens as the model size increases. Moreover, executing a forward pass for multiple tokens in parallel often takes nearly the same time as it does for just one token. These two observations lead to the development of speculative sampling, where a second smaller model is used to draft a few tokens, that are then validated or rejected using a single forward pass of the large model. Unfortunately, this method requires two models that share the same tokenizer and thus limits its adoption. As an alternative, we propose to use parallel decoding as a way to draft multiple tokens from a single model with no computational cost, nor the need for a second model. Our approach only requires an additional input token that marks the words that will be generated simultaneously. We show promising performance (up to 30% speed-up) while requiring only as few as $O(d_{emb})$ additional parameters.

1 Introduction

Since the Transformer architecture was proposed (Vaswani et al. [2023]), large language models have achieved impressive results across natural language processing benchmarks (Brown et al. [2020]). However, these remarkable achievements were only made possible because of dramatic increases in the number of parameters or model sizes (Brown et al. [2020], Wei et al. [2022]), resulting in considerable memory requirements and greater processing times. This problem is further exacerbated by the fact that, at inference time, transformers are used auto-regressively: a new model call is needed for each generated token. This is especially problematic due to the memory-bandwidth cost of recurrently loading the model parameters and the past keys and values tensors (Shazeer [2019]).

Recently, several works (Chen et al. [2023], Leviathan et al. [2023]) have proposed to reduce inference time by leveraging a smaller model to approximate generation from a larger model at a faster pace. The small model produces a few potential tokens, and the larger model evaluates all of the tokens at once in a single forward step. Importantly, the generation quality of the original large model is guaranteed by the rejection scheme that keeps only tokens that are generated with an identical distribution than the large model (Chen et al. [2023], Leviathan et al. [2023]). While effective in practice, this approach requires to deploy simultaneously two models that share the same vocabulary, creating memory and running time bottlenecks.

An alternative solution is to directly leverage the large model to generate multiple tokens at once, instead of generating them auto-regressively. This solution, called parallel decoding, can be implemented as a masked language model (Ghazvininejad et al. [2019]) or by copying the encoder input in the decoder in the context of encoder-decoder architectures (Gu et al. [2018]). These solutions

[*]Work done while at Apple

3rd Workshop on Efficient Natural Language and Speech Processing (NeurIPS 2023).

have the advantage over speculative sampling of avoiding the need for a second model, but they require substantial changes to the Transformer architecture that make them not suitable as such for accelerating the decoding of a given pre-trained language model.

In this work, we propose to combine the best of both directions in a variant of the speculative sampling that we call Parallel Speculative Sampling (PaSS). The idea is to generate candidate tokens via parallel decoding by adding a small number of "look-ahead embeddings" and generate output for each of these additional embeddings. This solution does not require a second model, nor modifications to the large language model. By design, our approach also generates at each step at least one token auto-regressively, guaranteeing the same loss-less quality of generations as speculating sampling methods. The memory overhead of adding the extra embeddings is $O(d_{emb})$ new weights, that need to be trained. This is several orders of magnitude smaller than any small model added by existing speculative sampling solutions. The most similar works to ours are Stern et al. [2018] and Cai et al. [2023] where they add look-ahead classification heads instead of embeddings, leading to a worse memory overhead of $O(d_{emb}K)$ where K is the vocabulary size. Additionally, Stern et al. [2018] focus solely on greedy decoding, and Cai et al. [2023] do not guarantee a loss-less decoding. Similarly, Zhang et al. [2023] also drop the second model, but still decode auto-regressively.

2 Method

First, in section 2.1, we briefly review the existing speculative sampling algorithm, as introduced by Chen et al. [2023] and Leviathan et al. [2023]. We then introduce our approach in section 2.2.

2.1 Speculative Sampling

The goal of speculative sampling is to speed up the inference time of a target LLM. The core idea behind this algorithm is that it is significantly faster to compute a single forward pass on n tokens in parallel than n forward passes sequentially. To fulfil this objective, a second smaller and faster model, *the drafter*, is used to generate a candidate sequence of tokens. The length of the sequence is a hyper-parameter of the algorithm. The target LLM is then presented with all of the candidate tokens at once, in a single pass. The rejection scheme proposed by Chen et al. [2023] and Leviathan et al. [2023] guarantees that the distribution of the drafted tokens is the same as if they were generated in the first place by the target LLM. Additionally, due to the rejection scheme, one more token can be sampled after the sequence of accepted tokens from the logits gathered during the iteration model call. This ensures that, even if all draft tokens were rejected, the model call would still be of use. The steps are presented in detail in algorithm 2 of the Appendix.

2.2 Parallel Speculative Sampling

We propose a modified version of speculative sampling based on parallel decoding, that does not require a second model. The steps of our method are detailed in algorithm 1, but, importantly, each iteration of our algorithm requires two calls of the LLM:

- **Drafting phase**: we call the model once to produce multiple tokens simultaneously using parallel decoding through look-ahead embeddings (see Sec. 2.2.1). The first generated token is not part of the draft to match the distribution in case of rejection.
- **Validation phase**: we call the model a second time to validate the draft (see Sec. 2.1). We sample a new token at the end of the sequence of accepted tokens, with no new model call.

The key behind our algorithm is that every call to the LLM adds at least one token to the final sequence of generated tokens. This guarantees that the algorithm is at least as fast as generating from the LLM directly, and it also guarantees that we produce a correct sequence of tokens even in the case where additional tokens are rejected. On top of this, our algorithm can produce and accept at each iteration, multiple additional tokens, leading to a guaranteed speed up. Overall, the standard auto-regressive wall time of the target LLM is a lower bound to our method, while the upper bound is a speed-up of $(L+2)/2\times$, where L is the number of tokens generated in the drafting phase. Our approach leverages the fact that the time required to process a single token or a small sequence of tokens does not differ significantly. This is because auto-regressive generation is mostly memory-bound, and processing additional tokens is thus negligible.

Algorithm 1 Parallel Speculative Sampling (PaSS) with Parallel Look-ahead Embeddings

Given L look-ahead tokens $\texttt{[LA]}_1, \ldots, \texttt{[LA]}_L$ and minimum target sequence length T.
Given auto-regressive target model $q(.|.)$ and initial prompt sequence $x_0, \ldots, x_t$.
Initialise $n \leftarrow t$.
while $n < T$ **do**
In parallel, sample the next token x_{n+1} and L draft tokens $\tilde{x}_1, \ldots, \tilde{x}_L$:

$$x_{n+1} \sim q(x|x_1, \ldots, x_n),\ \tilde{x}_1 \sim q(x|x_1, \ldots, x_n, \texttt{[LA]}_1),\ \ldots,\ \tilde{x}_L \sim q(x|x_1, \ldots, x_n, \texttt{[LA]}_1, \ldots, \texttt{[LA]}_L)$$

Set $n \leftarrow n + 1$
In parallel, compute $L + 1$ sets of logits from drafts $\tilde{x}_1, \ldots, \tilde{x}_L$:

$$q(x|x_1, \ldots, x_n),\ q(x|x_1, \ldots, x_n, \tilde{x}_1),\ \ldots,\ q(x|x_1, \ldots, x_n, \tilde{x}_1, \ldots, \tilde{x}_L)$$

for $t = 1 : L$ **do**
Sample $r \sim U[0, 1]$ from a uniform distribution.
if $r < \min\left(1, \frac{q(\tilde{x}_t|x_1, \ldots, x_{n-1}, \ldots, x_{n+t-1})}{q(\tilde{x}_t|x_1, \ldots, x_{n-1}, \texttt{[LA]}_1, \ldots, \texttt{[LA]}_t)}\right)$ **then**
Set $x_{n+t} \leftarrow \tilde{x}_t$ and $n \leftarrow n + 1$
else
Sample

$$x_{n+t} \sim (q(x|x_1, \ldots, x_{n-1}, \ldots, x_{n+t-1}) - q(x|x_1, \ldots, x_{n-1}, \texttt{[LA]}_1, \ldots, \texttt{[LA]}_t))_+$$

and Exit for loop.
end if
end for
If all L tokens $x_{n+1}, \ldots, x_{n+L}$ are accepted, sample extra token $x_{n+L+1} \sim q(x|x_1, \ldots, x_{n+L})$ and set $n \leftarrow n + 1$.
end while

2.2.1 Look-ahead embeddings

The target LLM is not trained to predict multiple tokens at once, and expect as input, the previously generated token. In order to build this ability in the target LLM, we introduce "look ahead" tokens, $\texttt{[LA]}_i$ for each look ahead position i for 1 to L. A sequence of these tokens is added at the end of the input sequence and defines the number of steps that the model will predict ahead. In other words, we replace the original input sequence of tokens $(w_1, \ldots, w_T)$ by a sequence with L additional tokens, that is $(w_1, \ldots, w_T, \texttt{[LA]}_1, \ldots, \texttt{[LA]}_L)$. This sequence is then processed with a single forward pass of the target model and produces for each new position a token from the original dictionary, i.e., without the extra $\texttt{[LA]}_i$ tokens. This approach only requires learning the embeddings associated with the new tokens on a small training dataset and has a memory overhead of Ld_{emb} parameters.

3 Experiments

We test our method on two tasks: text and code completion. For each task, we use different non-overlapping datasets for the training of the look-ahead embeddings and the evaluation of our approach (Sec. 3.1). We also briefly describe baselines in Sec. 3.2 and report main results in Sec. 3.3. All the experiments are run with a re-implementation of the 7B LLaMa model (Touvron et al. [2023]).

3.1 Data

We use the 2023/02/20 English Wikipedia dump (Wikimedia Foundation) for text completion and the Python split of The Stack corpus (Kocetkov et al. [2022]) for code completion. We divide each dataset into training and test split. For the evaluation, we randomly sample 200 examples from the test split and use the 32 first tokens as prompts. The maximum length for the generation is fixed at 512 tokens for all our experiments. We use TOP-K SAMPLING, with $k = 10$ and a temperature of 0.8

Table 1: Time for generating a sequence of length 512 tokens, given a prompt of 32 tokens, as a function of temperature (left) and number of look-ahead tokens (right). We use 4 look-ahead tokens unless said otherwise. The results reported in the right table are on The Stack data.

	The Stack			Wikipedia		
Temperature	0.8	0.5	0.2	0.8	0.5	0.2
Auto-regressive	12.52	12.69	12.72	12.45	12.30	12.55
`[UNK]` look-ahead	12.25	12.43	12.26	12.30	12.16	11.88
PaSS	9.79	9.46	8.96	10.23	9.78	9.43

# LA tokens	Time
2	10.03
4	9.79
6	9.66
8	9.94

unless said otherwise. For code completion, we also evaluate on the HumanEval benchmark [Chen et al., 2021], to validate that, as expected, our algorithm does not degrade the quality of generation.

3.2 Baselines

We compare our method with two baselines. **Autoregressive generation:** this baseline consists of autoregressively generating tokens from the LLM. We sample one token at a time using the KV cache for speedup. `[UNK]` **as look-ahead token:** we apply our method with a fixed `[UNK]` token instead of trained look-ahead embeddings. We use the KV cache and update it after every model call according to the number of drafted tokens and the number of accepted tokens.

3.3 Results

Impact for different sampling schemes. On the left panel of Table 1, we compare the running time of our approach with the two baselines for different sampling schemes. We vary the temperature of the sampler from high variance in the generations (high temperature) or low variance (low temperature). As expected, the speed-up is more important for lower temperatures, where the distribution of tokens is more peaky and easier to predict with an approximated scheme like PaSS. We observe almost no gain compared to auto-regressive generation when using `[UNK]`. Compared to the speed-up provided by PaSS, this shows that the finetuning of the look-ahead embeddings captures important information to predict future tokens.

Impact of the number of look ahead embeddings. On the right panel of Table 1, we measure the impact of the number of look-head steps on our approach. Running time decreases steadily up to 6 look-ahead steps, but more look-ahead steps annihilate the benefits of this approach.

Impact of PaSS on final performance. Finally, in Table 2, we confirm that our decoding does not impact the performance of the model on 2 different generating tasks. We only observe changes in performance that are below the margin of error, while improving the running time by up to 30%.

Table 2: Average time for generating one completion on the HumanEval dataset, as well as the PASS@N metric. Following previous work, we use a temperature of 0.1 for PASS@1 and a temperature of 0.8 for PASS@10. We use $k = 25$ for PASS@10. For PaSS, we use 4 look-ahead tokens.

	PASS@1		PASS@10	
	Time	Perf.	Time	Perf.
Auto-regressive	10.52 sec	13.2 %	10.15 sec	22.5 %
PaSS	7.17 sec	13.4 %	8.17 sec	22.5 %

4 Conclusion

We presented the parallel speculative sampling (PaSS) algorithm, a variant of the speculative sampling algorithm that does not require a second draft model: tokens are drafted in parallel through the use of masked-decoding via fine-tuned look-ahead embeddings. We showed that our method achieves significant speed-ups (up to 30%) by only learning as little as $O(d_{emb})$ additional weights. In future work, we want to explore how to improve the quality of parallel generation with look-ahead tokens, as we believe this is the most promising direction to improve performance of the PaSS algorithm.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, and Tri Dao. Medusa: Simple framework for accelerating llm generation with multiple decoding heads. `https://github.com/FasterDecoding/Medusa`, 2023.

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. Mask-predict: Parallel decoding of conditional masked language models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6112–6121, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1633. URL `https://aclanthology.org/D19-1633`.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. Non-autoregressive neural machine translation, 2018.

Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, and Harm de Vries. The stack: 3 tb of permissively licensed source code. *Preprint*, 2022.

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

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

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models, 2018.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

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

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models, 2022.

Wikimedia Foundation. Wikimedia downloads. URL `https://dumps.wikimedia.org`.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft verify: Lossless large language model acceleration via self-speculative decoding, 2023.

5 Supplementary Material

5.1 Speculative Sampling Algorithm

Algorithm 2 Speculative Sampling (SpS) with Auto-Regressive Target and Draft Models

Given look-ahead L and minimum target sequence length T.
Given auto-regressive target model $q(.|.)$, auto-regressive draft model $p(.|.)$ and initial prompt sequence $x_0, \ldots, x_t$.
Initialise $n \leftarrow t$.
while $n < T$ **do**
  for $t = 1 : L$ **do**
    Sample draft auto-regressively $\tilde{x}_t \sim p(x|x_1, \ldots, x_n, \tilde{x}_1, \ldots, \tilde{x}_{t-1})$
  end for
  In parallel, compute $L + 1$ sets of logits from drafts $\tilde{x}_1$, ..., $\tilde{x}_L$:
    $q(x|x_1, \ldots, x_n),\ q(x|x_1, \ldots, x_n, \tilde{x}_1),\ \ldots,\ q(x|x_1, \ldots, x_n, \tilde{x}_1, \ldots, \tilde{x}_L)$
  for $t = 1 : L$ **do**
    Sample $r \sim U[0, 1]$ from a uniform distribution.
    if $r < \min\left(1, \frac{q(\tilde{x}_t|x_1,\ldots,x_{n+t-1})}{p(\tilde{x}_t|x_1,\ldots,x_{n+t-1})}\right)$ **then**
      Set $x_{n+t} \leftarrow \tilde{x}_t$ and $n \leftarrow n + 1$
    else
      Sample $x_{n+t} \sim (q(x|x_1, \ldots, x_{n+t-1}) - p(x|x_1, \ldots, x_{n+t-1}))_+$ and exit for loop.
    end if
  end for
  If all tokens $x_{n+1}, \ldots, x_{n+L}$ are accepted, sample extra token $x_{n+L+1} \sim q(x|x_1, \ldots, x_{n+L})$ and set $n \leftarrow n + 1$.
end while

5.2 Training details

For all our trainings (and evaluations), we load models in bfloat16. We freeze all the models parameters except for the new embeddings. For each batch, we randomly select a position where to insert the look-ahead embeddings and compute the loss only on the corresponding outputs. Before training, we initialize the new embeddings with the *UNK* token embedding. We use the AdamW optimizer, with a learning rate of 0.01 and a batch size of 8 sequences. We train for 10k additional steps, with 2k warmup steps and a cosine learning rate schedule.