Generation Meets Verification: Accelerating Large Language Model Inference with Smart Parallel Auto-Correct Decoding

Anonymous ACL submission

Abstract

This research aims to accelerate the inference speed of large language models (LLMs) with billions of parameters. We propose **S**mart **P**arallel **A**uto-**C**orrect d**E**coding (SPACE), an innovative approach designed for achieving lossless acceleration of LLMs. By integrating semi-autoregressive inference and speculative decoding capabilities, SPACE uniquely enables autoregressive LLMs to parallelize token generation and verification. This is realized through a specialized semi-autoregressive supervised fine-tuning process that equips existing LLMs with the ability to simultaneously predict multiple tokens. Additionally, an auto-correct decoding algorithm facilitates the simultaneous generation and verification of token sequences within a single model invocation. Through extensive experiments on a range of LLMs, SPACE has demonstrated inference speedup ranging from 2.7x-4.0x on HumanEval-X while maintaining output quality.

1 Introduction

The majority of current large language models (LLMs), including prominent examples like ChatGPT (Brown et al., 2020) and LLaMA (Touvron et al., 2023), are autoregressive (AR) in nature. During the inference stage, these AR models generate tokens one by one in a sequential manner. This sequential approach limits parallelism, leading to underutilization of modern parallel computing resources such as graphics processing units (GPUs). Consequently, there is a noticeable increase in latency during the inference stage. This issue becomes more pronounced when dealing with advanced LLMs, typically equipped with billions of parameters, where speed is crucial but hindered by the sequential token generation mechanism.

A straightforward method to mitigate the latency is to adapt the model to predict multiple future tokens in parallel. Such models are commonly referred to as semi-autoregressive (SAR) models (Xiao et al., 2023). Nonetheless, the vast majority of LLMs are inherently AR and, hence, unable to perform inference in a SAR manner. In addition, SAR models commonly experience a deterioration in the quality of output due to their parallel decoding nature (Xiao et al., 2023). Furthermore, it is worth mentioning that pretraining a SAR LLM from scratch incurs significant computational expenses.

Another effective ways to speed up AR sampling is speculative decoding (Leviathan et al., 2023; Chen et al., 2023; Miao et al., 2023). Speculative decoding typically adheres to the 'draft-then-verify' paradigm, wherein multiple candidate tokens are initially generated by fast-to-infer smaller models, and are subsequently validated in parallel by the larger LLM. This validation process, based on rejection sampling, ensures that the final output is consistent with the LLM's distribution, thereby achieving lossless speedup. Nonetheless, speculative decoding is contingent on the availability of smaller models, which are challenging to procure. Further, these additional models incur extra memory overhead during inference.

Integrating SAR inference with speculative decoding presents a promising approach to accelerate language model inference. By adapting a model to autonomously generate and validate a sequence of future tokens, we establish an efficient and self-reliant process that greatly enhances the speed of inference. This union yields substantial practical benefits: it eliminates the requirement for smaller auxiliary models, thereby simplifying the overall implementation and reducing memory overhead during inference. Furthermore, by shifting the emphasis away from precise prediction of multiple tokens towards speculative generation followed by verification, the difficulty of the SAR training phase can be significantly reduced.

In this paper, we propose Smart Parallel Auto-Correct dEcoding (SPACE), an innovative ap-

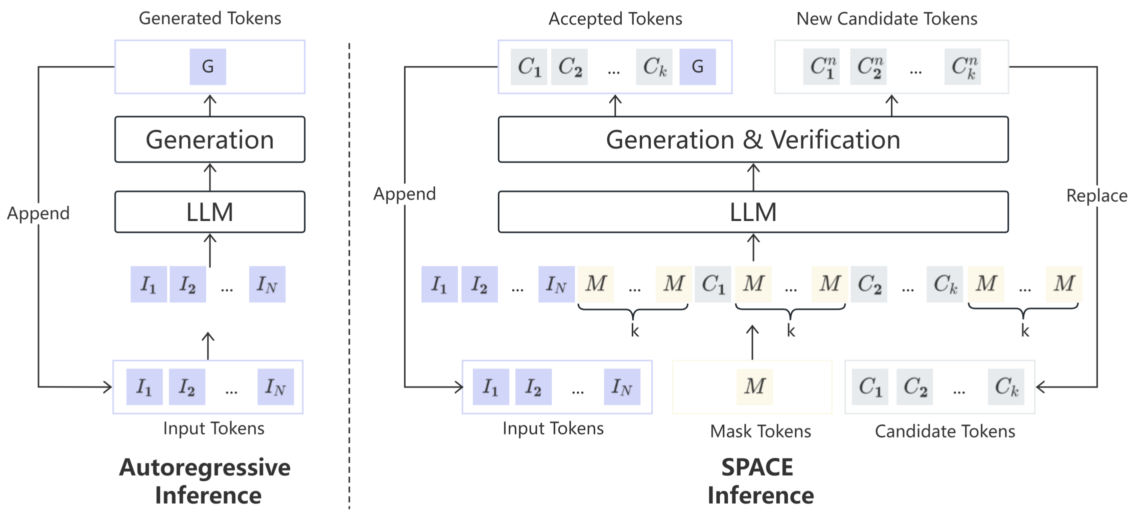

Figure 1: A visual comparison between conventional AR inference (left) and SPACE inference (right) is illustrated. In AR inference, token generation proceeds in a sequential manner, with only one token output per decoding step. In SPACE inference, the input token sequence is augmented with $k + 1$ groups of mask tokens and k candidate tokens. The candidate tokens undergo verification, and k new candidate tokens are generated from one of the mask groups after a single model invocation. SPACE allows for a variable number of tokens to be generated in each step, with the quantity ranging from a minimum of 1 to a maximum of $k + 1$.

proach that allows LLMs to generate multiple tokens speculatively while simultaneously verifying them. SPACE harmonizes a SAR model with a draft-then-verify inference algorithm to optimize inference speed while maintaining high model quality. We demonstrate that a pretrained AR language model can be adapted to produce probable token sequences in parallel through semi-autoregressive supervised fine-tuning (SAR-SFT). This strategy obviates the need for supplementary smaller models and maintains the fine-tuning process within reasonable computational demands. We also introduce an auto-correct decoding algorithm that enables the generation and validation of token candidates to occur concurrently within a single invocation of a model, thereby significantly boosting inferential efficiency. A visual comparison between traditional AR inference and SPACE inference mechanism can be found in Figure 1. Our key contributions are summarized as follows:

- We propose a semi-autoregressive supervised fine-tuning scheme that empowers autoregressive LLMs to generate multiple tokens at once, without requiring substantial computational overhead.

- We pioneer an auto-correct decoding algorithm that facilitates the concurrent generation and validation of token candidates within a single forward pass of the model, ensuring a lossless acceleration in inference speed.

- Our extensive experiments, conducted across various LLMs with parameters ranging from 6B to 70B, validate that SPACE is effective in achieving an inference speedup from 2.7x to 4.0x in HumanEval-X while maintaining output quality.

2 Related Work

Speculative Decoding Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) aims to accelerate LLM decoding by utilizing a smaller draft model to anticipate the larger target model's outputs. These predictions are then validated by the target model. The success of speculative decoding heavily relies on the precision of the draft model's predictions. To enhance accuracy, researchers have adopted various strategies such as employing ensembles of boosted draft models (Miao et al., 2023), staged draft models (Spector and Re, 2023), retraining the target model with addition of auxiliary prediction heads (Stern et al., 2018), introducing advanced coordination policies (Kim et al., 2023) and refining the decoding algorithm (Sun et al., 2023). However, speculative decoding hinges on the accessibility of suitable smaller models, which can be difficult to obtain and often requiring extra training and careful tuning (Liu et al., 2023). SPACE circumvents this challenge by fine-tuning the target model to prognosticate future token sequences in parallel, eliminating the dependency on extra small model.

Recent advancements like Lookahead Decoding (Fu et al., 2023) and Self-Speculative (Zhang et al., 2023) suggest innovative approaches where a single LLM is employed for both generating and verifying tokens. These methods are appealing due to simplifying the process by not necessitating multiple models or additional training regimes. However, it's noteworthy that despite these methodologies being more streamlined, they typically achieve a lower speedup ratio when compared to SPACE.

Semi-Autoregressive Decoding SAR departs from the conventional AR approach by decoding multiple tokens in parallel, thereby significantly enhancing inference efficiency. Particularly in machine translation, SAR has achieved a fivefold speed increase while preserving 88% of the model quality (Wang et al., 2018). Recent research efforts to enhance SAR performance in machine translation include employing alignment-focused training objectives (Gu and Kong, 2021), innovating model architectures (Huang et al., 2022), etc. However, to the best of our knowledge, exploration of SAR in conjunction with decoder-only LLMs remains limited. A recent study that aligns closely with our work is SpecDec (Xia et al., 2022), which employs a strategy of initially decoding a block of tokens quickly as a draft with a SAR model before refining this draft using an AR model. However, a notable difference is that SpecDec requires the training of an extra SAR model, which introduces resource overhead.

3 Methods

SPACE primarily comprises two components: the SAR-SFT mechanism and the auto-correct decoding algorithm. The SAR-SFT mechanism enhances an autoregressive LLM's capacity for speculative multi-token generation in a single decoding step. Meanwhile, the auto-correct decoding algorithm allows the LLM to concurrently generate and verify candidate tokens. We introduce the details of these two components in the following subsections.

3.1 Semi-Autoregressive Supervised Finetuning

Conventionally a pretrained LLM undergoes a process known as supervised fine-tuning (SFT) to align its output with human instructions. Specifically, given the prompt token sequence X and the answer token sequence $Y = \{y_1, y_2, \cdots, y_N\}$, the AR model is trained in SFT with loss function

$$\mathcal{L}_{AR} = -\sum_{t=1}^{N} \log P(y_t|y_{<t}, X; \theta), \quad (1)$$

where y_t is the token to be predicted at time step t, $y_{<t}$ is the tokens predicted in previous $t-1$ decoding steps and θ is the model parameters.

In the proposed SAR-SFT scheme, our objective is to train the model to generate k consecutive tokens when presented with an input sequence containing k mask tokens. To achieve this, we employ an autoregressive loss $\mathcal{L}_{AR}$ with a probability p_{ar}. Conversely, with a complementary probability of $1-p_{ar}$, we randomly sample an index m from $\{0, 1, \cdots, N-k\}$ and obtain $y_{<m}$ from the answer token sequence. Subsequently, we append k mask tokens "[M]" to $y_{<m}$ to form $y^k_{<m}$:

$$y^k_{<m} = \{y_1, y_2, \cdots, y_{m-1}, \underbrace{[M], \cdots, [M]}_{\times k}\}. \quad (2)$$

The model is trained with the SAR loss function defined as follows:

$$\begin{aligned}\mathcal{L}_{SAR} = &-\sum_{t=1}^{m} logP(y_t|y_{<t}, X; \theta) \\ &-\sum_{t=m}^{m+k} logP(y_t|y^k_{<m}, X; \theta) \quad (3)\end{aligned}$$

The final loss function we used in SAR-SFT is

$$\mathcal{L} = p_{ar}\mathcal{L}_{AR} + (1-p_{ar})\mathcal{L}_{SAR} \quad (4)$$

Intuitively, the parameters p_{ar} plays a critical role in striking a balance between the AR loss and the SAR loss. By selecting an appropriate value for p_{ar}, the LLM is trained not only to adhere to instructions but also to predict multiple tokens at each decoding step.

We note that the primary goal of SAR-SFT is not to compel the LLM to predict several tokens in parallel with high accuracy, as this can be an exceedingly challenging task that significantly increases training costs. Rather, our goal is to enable the LLM to make an "educated guess" about the upcoming few tokens, which is more attainable. This strategy allows the model to improve its inference efficiency by preparing probable token sequences beforehand, which can later be validated and refined by the auto-correct decoding algorithm introduced in next subsection.

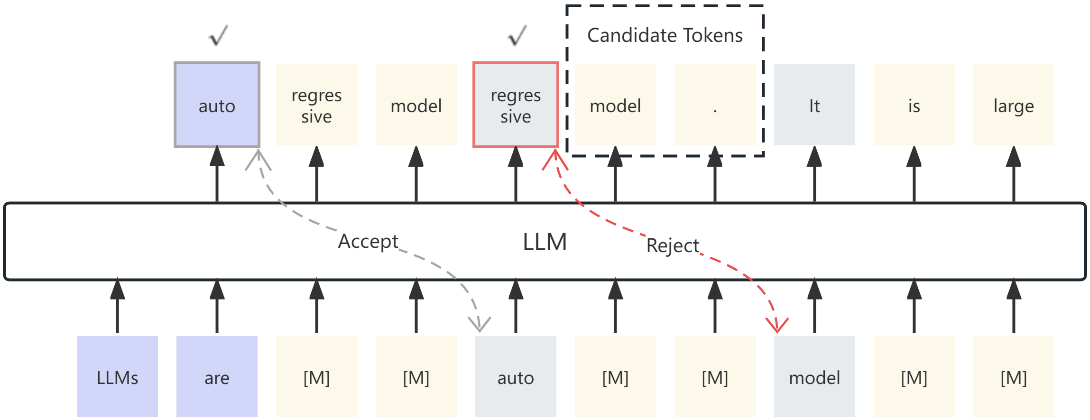

Figure 2: An illustrative example of the auto-correct decoding step in SPACE. In this example, the first candidate token "auto" is accepted, while the second candidate token "model" is rejected. In this case, the LLM generates two new tokens "auto" and "regressive" in this decoding step and two new candidate tokens "model" and ".".

3.2 Auto-Correct Decoding Algorithm

Unlike previous methods (Leviathan et al., 2023; Chen et al., 2023; Miao et al., 2023) that rely on auxiliary models, our approach SPACE streamlines the process by using the same LLM for both generation and subsequent verification of candidate tokens. To enhance inference efficiency, we have developed an algorithm that enables this unified LLM to concurrently verify tokens from the current step and generate new candidates for the next step within a single forward pass.

Algorithm 1 outlines the auto-correct decoding algorithm used in SPACE and Figure 2 gives an illustrative example. We note that this decoding algorithm is applicable to both greedy and random sampling settings. Since greedy sampling can be considered a special case of random sampling, we introduce SPACE within the broader context of random sampling setting without loss of generality.

Given a sequence of input tokens $\mathcal{T} = \{x_1, x_2, \cdots, x_l\}$ and a list of k candidate tokens $L_c = \{c_1, c_2, \cdots, c_k\}$ generated from previous step, we first construct a sequence of input tokens $\mathcal{I}$ as follows:

$$\begin{aligned} \mathcal{I} =& \{x_1, x_2, \cdots, x_l, \\ & L_m^k, c_1, L_m^k, c_2, \cdots, c_k, L_m^k\}, \end{aligned} \tag{5}$$

where $L_m^k = \underbrace{[M], \cdots, [M]}_{\times k}$ represents a group of k mask tokens and there are $k+1$ groups of them in $\mathcal{I}$. The input token sequence is expanded by $k \cdot (k+2)$ additional tokens, resulting in a total length of $|\mathcal{I}| = l + k \cdot (k+2)$. These $k+1$ groups

Algorithm 1 The auto-correct decoding algorithm

Input: A sequence of input tokens $\mathcal{T}$, number of mask tokens k, large language model $\mathcal{M}$
Output: A sequence of generated tokens $\mathcal{O}$

```
O = T, L_c = [0] × k, P_c = [+∞] × k
while True do
    l = len(O)
    Get I, Ā, P̄ according to equation (5)-(7)
    P = M(I, Ā, P̄)              ▷ Get the output logits
    idx = l + 1
    Q = P[l]                    ▷ The logit of the l-th token
    for i = 1 to k do
        r ~ U(0, 1)
        if r ≤ Q(L_c[i])/P_c[i] then
            O.append(L_c[i])
            idx = idx + k + 1
            Q = P[l + i * (k + 1)]
        else
            break
        end if
    end for
    a ~ Q                       ▷ Sample one extra token
    O.append(a)
    if <EOS> in O then
        return O[:eos_index]
    end if
    L_c ~ P[idx : idx + k]      ▷ New candidates
    P_c = P[idx : idx + k](L_c) ▷ Probability
end while
```

of mask tokens are designated for the generation of new candidate tokens. Depending on the number of accepted tokens, the generation results from one of the mask token groups will be chosen as candidate tokens, with further details to be presented later.

Since LLM decoding is primarily bounded by memory bandwidth, we can merge the generation and verification in the same forward step, leveraging GPU's parallel processing power to hide overheads. We achieve this by designing special attention mask $\bar{A} \in \{0,1\}^{|\mathcal{I}|\times|\mathcal{I}|}$ and positional encoding $\bar{P} \in \mathcal{N}^{|\mathcal{I}|}$ as follow:

$$\bar{A}_{ij} = \begin{cases} 1 & i \geq j, \mathcal{I}[j] \neq M \\ 1 & i \geq j, i-j < k, \mathcal{I}[i] = \mathcal{I}[j] = M \\ 0 & otherwise \end{cases} \tag{6}$$

$$\bar{P}_i = \sum_{j=1}^{|\mathcal{I}|} A_{ij} - 1 \tag{7}$$

The attention mask is tailored such that masked tokens can causally attend only to other mask tokens within the same group and to preceding non-masked tokens. Furthermore, all non-masked tokens are restricted to causally attend to prior non-masked tokens, and are unable to attend to any preceding masked tokens. An illustrative example of the attention mask configuration is depicted in Figure 3 with $k = 2$.

	LLMs	are	[M]	[M]	auto	[M]	[M]	model	[M]	[M]
LLMs	1									
are	1	1								
[M]	1	1	1							
[M]	1	1	1	1						
auto	1	1			1					
[M]	1	1			1	1				
[M]	1	1			1	1	1			
model	1	1			1			1		
[M]	1	1			1			1	1	
[M]	1	1			1			1	1	1

Figure 3: An illustrative example of the attention mask used in SPACE. In this example, $k = 2$ and the input is extended with 8 tokens. "LLMs are" are the input query, "auto" and "model" are two candidate tokens that need to be verified.

The extended input token sequence $\mathcal{I}$, together with attention mask $\bar{A}$ and positional encoding $\bar{P}$, are passed through LLM. This facilitates the inference process, allowing the LLM to generate the normalized output logits, denoted as P, as outlined in line 5 of Algorithm 1.

The candidate tokens are verified through reject sampling, which is detailed from line 6 to line 22 in Algorithm 1. Denote P_c as the list of semi-autoregressive probability of candidate tokens obtained from previous step. Formally $P_c[i]$ is defined as:

$$P_c[i] = P(c_i|x_1, \cdots, x_{l-1}, \underbrace{[M], \cdots, [M]}_{\times i}) \tag{8}$$

Denote Q_c as the list of autoregressive probability of candidate tokens obtained from current step [1].

$$Q_c[i] = P(c_i|x_1, \cdots, x_l, c_1, \cdots, c_{i-1}) \tag{9}$$

Starting from $i = 1$, we accept token c_i with probability:

$$\min(1, \frac{Q_c[i]}{P_c[i]}) \tag{10}$$

This can be implemented by first sample a random number uniformly from $[0, 1]$, and then accept the token if this random number does not exceed the ratio $Q_c[i]/P_c[i]$. Upon acceptance of token c_i, the algorithm output c_i and proceeds to validate the subsequent token c_{i+1} using the same criterion; conversely, if c_i is rejected, the verification process terminates immediately. It is important to observe that during each decoding step, the number of generated tokens ranges from a minimum of one to a maximum of $k + 1$. By employing reject sampling, it can be proved that the distribution of the output token sequence matches that of the AR inference process in the LLM. For a more comprehensive explanation of this claim, readers can refer to prior research (Leviathan et al., 2023; Chen et al., 2023).

In the case that there are i^* accepted candidate tokens, where $0 \leq i^* \leq k$, the generation of new candidate tokens for the subsequent step commences from the $(i^* + 1)$-th mask token group, as denoted in lines 23-24 of Algorithm 1. This approach ensures the generation of k candidate tokens at each decoding step, as illustrated in Figure 2.

[1] By definition, $Q_c[i]$ is equivalent to $Q(L_c[i])$ in line 10 of Algorithm 1.

4 Experiments

4.1 Experimental Settings

Training We conduct experiments on LLMs with various sizes, including ChatGLM3-6B-Base (Du et al., 2022), LLaMA-2 (7B, 13B, 70B) (Touvron et al., 2023), Qwen-14B (Bai et al., 2023), InternLM-20B (Team, 2023), Falcon-40B (Almazrouei et al., 2023). To ensure reproducibility, we finetune the models using publicly available SFT datasets including Alpaca-GPT4 (Peng et al., 2023), Lima (Zhou et al., 2023), Oaast-SFT (LAION-AI, 2023), CodeAlpaca (Chaudhary, 2023), and OpenPlatypus (Lee et al., 2023). The details of these dataset are listed in Table 3 in Appendix A.1. There are in total 166,993 training samples. We add the mask token as a special token and initialize its embedding with normal distribution. Unless otherwise specified, we set the number of mask tokens $k = 5$ and $p_{ar} = 0.5$. We finetune the models for 2 epochs with a learning rate as 5e-5. The training details can be found in Appendix A.1.

Inference In our assessment of SPACE, we employ four distinct datasets: Chatbot Instruction Prompt (CIP) (Palla, 2023), MT-Bench (Zheng et al., 2023a), HumanEval-X (Zheng et al., 2023b) and XSum (Narayan et al., 2018). CIP is a conversational dataset from which we utilize prompts to simulate realistic conversations. MT-Bench is a dataset comprised of multi-turn questions, encompassing a wide range of topics. HumanEval-X is a standard benchmark for Python code generation and Pass@10 is used as the metric. Lastly, the XSum dataset, which tasks models with summary generation, is evaluated using ROUGE-L.

For inference baseline, we adopt the generation algorithm provided by the Huggingface Transformers library (Wolf et al., 2020), executing it in an autoregressive fashion. We conduct the experiments on a server with eight A800 (80GB) GPUs. By default, we set the batch size to 1 during inference. To evaluate the inference efficiency of SPACE, we employ two metrics: speedup and average accepted tokens. The speedup metric is defined as the ratio of the inference speed of the baseline method (measured in tokens per second) to the inference speed achieved using SPACE. The second metric, average accepted tokens, is computed as the ratio of the total number of tokens generated to the number of inference steps performed by the LLM. The evalutation details can be found in Appendix A.2.

4.2 Experimental Results

4.2.1 Inference Efficiency

The experimental results on XSum, HumanEval-X and CIP under greedy sampling setting are shown in Table 1. Under greedy decoding conditions, we anticipate identical outputs from models applying SPACE and baseline autoregressive generation method. However, the results exhibit occasional discrepancies between the two, which can be attributed to numerical variations during decoding that cause the generation of different tokens, potentially leading to significant divergence in the resulting sequences. Despite these observed differences, SPACE predominantly corresponds closely with baseline performance levels in both the XSum and HumanEval-X benchmarks. Moreover, SPACE demonstrably realizes a speedup in the range of 1.5 to 4.0, depending on the models and datasets. The maximal acceleration, seen in LLaMA-2-70B on HumanEval-X, clocks in at an impressive 4.01. More experimental results of SPACE under random sampling can be found in Appendix A.3.

From the aboved results, we have the following three observations: First, as compared to the autoregressive inference baseline, SPACE delivers lossless speedup when applied to models of varying sizes, showcasing its broad applicability. Specifically, the results attained by SPACE in tasks such as XSum and HumanEval-X closely mirror those achieved by the baseline method, as indicated by the comparable performance metrics listed in parentheses in Table 1.

Second, the magnitude of speedup experienced is model-specific, indicating that the efficiency benefits of SPACE can differ in models. This variance might stem from several factors: (1) the models' vocabularies vary, with less efficient vocabularies possibly leading to greater predictability and thus higher speedup; and (2) models with more parameters often enjoy more substantial speedup, likely owing to their superior predictive capabilities that facilitate earlier anticipation of forthcoming tokens.

Lastly, when applying SPACE to different tasks, the same model can exhibit dramatically different speedup ratios. In particular, tasks that involve programming, such as those in the HumanEval-X benchmark, exhibit the most significant speedup, achieving an average rate of 3.33 using greedy sampling. This observation aligns with the results in previous research (Chen et al., 2023), and could be attributed to the inherently structured and pre-

Model	XSum			HumanEval-X			CIP	
	ROUGE-L	Avg. Tokens	Speed-up	Pass@10	Avg. Tokens	Speed-up	Avg. Tokens	Speed-up
ChatGLM-3-6B	14.5 (14.4)	2.04	1.48	18.3 (18.3)	3.34	2.73	1.80	1.50
LLaMA-2-7B	16.0 (15.9)	2.23	1.94	18.9 (18.9)	3.54	3.18	1.85	1.71
LLaMA-2-13B	15.1 (15.1)	2.36	2.10	20.1 (20.1)	3.76	3.44	1.99	1.81
Qwen-14B	17.2 (17.2)	2.15	1.94	26.8 (26.8)	3.51	3.18	1.85	1.70
InternLM-20B	16.4 (16.3)	2.15	1.96	21.3 (21.3)	3.31	3.19	1.80	1.64
Falcon-40B	15.7 (15.7)	2.17	2.01	20.7 (20.7)	3.58	3.61	1.96	2.03
LLaMA-2-70B	16.4 (16.5)	2.54	2.35	28.0 (28.0)	4.32	4.01	2.09	1.91

Table 1: The experimental results on XSum, HumanEval-X and CIP under greedy sampling setting. We show the average accepted tokens (Avg. Tokens) and inference speedup (Speedup) for each datasets. The number in parentheses shows the corresponding results of the baseline method.

dictable nature of programming code.

4.2.2 Impact of SAR-SFT on Model Quality

While SPACE accelerates inference speed, it is imperative to explore whether LLMs trained with SAR-SFT suffer performance degradation compared to those trained with the conventional SFT approach. To this end, we train LLMs with SFT under the same datasets and training configuration used for SAR-SFT. Note that by setting $p_{ar} = 1$, SAR-SFT effectively becomes equivalent to SFT.

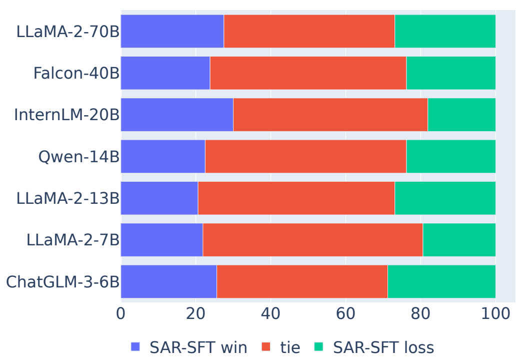

Figure 4: Win rate comparison in MT-Bench: SAR-SFT versus SFT judged by GPT-4. Best viewed in color.

For a comprehensive comparison, MT-Bench was employed with GPT-4 serving as the evaluator to measure the performance disparity between the LLMs trained with the two training schemes. The results are presented in Figure 4. We can observe that models trained with SAR-SFT scheme have comparable performance as compared to their SFT counterparts. Specifically, the majority of questions assessed in MT-Bench ended in a deadlock across all models, implying that training an LLM with

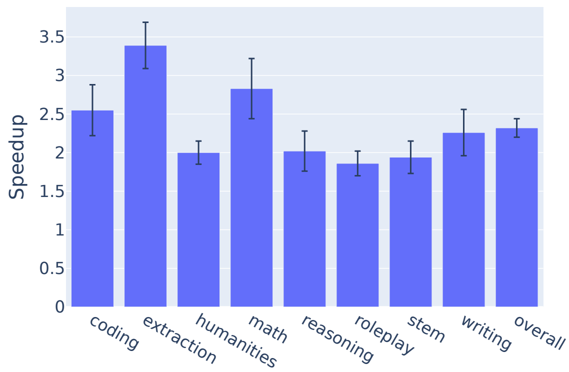

Figure 5: The mean and standard deviation of speedup for all models under greedy sampling setting in MT-Bench.

SAR-SFT does not deteriorate the model's quality. Additionally, SAR-SFT-trained models have exhibited advantages in speed. The mean and standard deviation of the speedup for all models in various tasks within MT-Bench are shown in Figure 5. It becomes evident that the speedup ratios vary considerably across different tasks, with the highest gains observed in tasks related to extraction, math, and coding. On averaged, all the models achieved a speedup ratio of 2.3 in MT-Bench dataset. More details can be found in Table 4 in the appendix.

To further validate that SAR-SFT does not compromise the model's effectiveness, a comprehensive evaluation was conducted using a suite of widely adopted benchmarks, including MMLU, BoolQ, and others. More detailed can be found in Appendix A.4.

4.2.3 Ablation Study

Our ablation study investigates the impact of varying the number of masked tokens, denoted as k, on

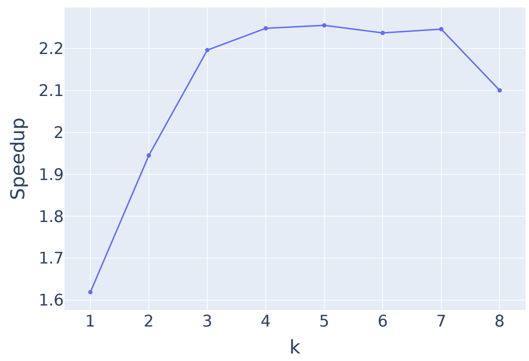

Figure 6: Ablation study on number of mask tokens based on LLaMA-2-7B. The speedup are evaluated under greedy sampling setting on MT-Bench dataset.

the speedup ratio of the LLaMA-2-7B model using the MT-Bench dataset. The results of this analysis are presented in Figure 6. Our findings indicate that a setting of $k = 5$ achieves an optimal balance for the model's performance. During the SAR-SFT phase, the LLM is tasked with concurrently predicting a sequence of k subsequent tokens. Increasing the value of k elevates the complexity of the prediction task and introduces computational overhead during inference, which may inversely correlate with the acceleration of the decoding process. Conversely, setting too low a value for k leads to an underutilization of the model's capacity for parallel decoding, potentially resulting in a less pronounced improvement in decoding speed.

4.3 Integration with TGI

When deploying LLMs for production use, it's common to leverage advanced LLM serving engines designed to enhance the efficiency of text generation tasks. The Text Generation Inference (TGI) (HuggingFace, 2023) framework is one such example, widely recognized for its support of a suite of acceleration techniques. TGI typically implements methods like flash attention, tensor parallelism, and continuous batching, among others, to enhance the speed of LLM inference.

We have integrated SPACE with the TGI framework. The primary objective of this integration is to ascertain whether SPACE can yield acceleration gains even when combined with other advanced inference-optimizing techniques presented in TGI. To quantitatively measure the inferring speedup provided by SPACE when integrated with TGI, we have carried out a thorough comparison, and the results are shown in Figure 7. The results were encouraging: with SPACE, TGI achieved a speed increase ranging from 1.5x to 3.4x across various model sizes. Remarkably, the incorporation of SPACE enabled the 13 billion-parameter LLaMA model to reach inference speeds comparable to, if not surpassing, those of a 7 billion-parameter model without SPACE supports. We will release our implementation of TGI with SPACE once the paper is accepted.

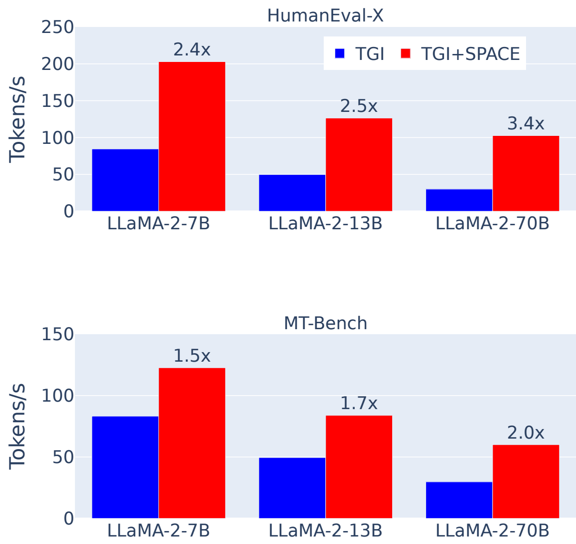

Figure 7: Token generation speed (Tokens/s) and speedup for LLaMA-2 (7B, 13B, 70B) with TGI and SPACE integration on HumanEval-X and MT-Bench datasets under greedy sampling setting. Best viewed in color.

5 Conclusion

In this paper, we introduce SPACE, an innovative approach designed to accelerate the inference speed of LLMs. SPACE seamlessly incorporates a semi-autoregressive model with a novel draft-then-verify inference algorithm. Our experiments reveal that an autoregressive LLM, fine-tuned in a semi-autoregressive approach, can generate likely sequences of tokens in parallel. The adoption of an effective auto-correct decoding algorithm facilitates the simultaneous generation and verification of token sequences. Experimental results on various LLMs show SPACE can achieve 2.7x-4.0x speedup on HumanEval-X while still preserving model quality.

6 Limitations

While SPACE has demonstrated potential in accelerating the inference of LLMs, it also brings about certain limitations that must be acknowledged: First, the primary advantage offered by SPACE is the acceleration of the inference process through the introduction of additional input tokens during decoding, which has the potential to reduce the number of forward passes that LLMs require. However, the presence of these additional tokens inevitably leads to increased computation overhead, notably in terms of FLOPs, when compared to conventional autoregressive decoding. Therefore, it becomes crucial to conduct an exhaustive study on the energy consumption of methods like SPACE, to fully understand and mitigate their ecological impact. The sustainability of deploying such acceleration techniques, considering long-term environmental implications, must factor into the development of responsible AI technologies.

Furthermore, it is important to recognize that the gain in inference speed facilitated by SPACE is variable across different tasks. Our empirical observations suggest that the speedup is inconsistent, and the limited datasets examined in this study could contribute to skewed outcomes. Besides, our evaluations for SPACE were conducted exclusively on English datasets; consequently, the extent to which SPACE can accelerate inference in other languages has not yet been investigated. It is plausible that there are specific datasets where SPACE exhibits a significantly lower degree of acceleration—a scenario not captured within the confines of our experimental array.

Moreover, we do not compare SPACE with other inference-accelerating methods in this paper. The lack of a standardized benchmark combined with the potential variability introduced by different model architectures, evaluation datasets, and hardware configurations makes such comparisons challenging. Rather than drawing indirect comparisons based on the speedup ratios reported from previous work, we aim to provide a more equitable evaluation by reproducing selected existing methods and assessing them using an identical setup in our future work.

Lastly, we leveraged MT-Bench along with a collection of well-established benchmarks, such as MMLU, PIQA, AGIEval, and others, to gauge model performance when trained with SAR-SFT as opposed to traditional SFT methodologies. Despite this extensive set of evaluations, it is critical to emphasize that benchmarking the comprehensive capabilities of LLMs remains a challenge, and the datasets engaged in this research fall short of enabling a definitive judgment. To this end, we advocate for the application of SPACE in diverse downstream tasks by the research community, which will offer a more rounded understanding of its practical utility and limitations.

References

Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Merouane Debbah, Etienne Goffinet, Daniel Heslow, Julien Launay, Quentin Malartic, et al. 2023. Falcon-40b: an open large language model with state-of-the-art performance. *Findings of the Association for Computational Linguistics: ACL*, 2023:10755–10773.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. 2009. The fifth pascal recognizing textual entailment challenge. In *TAC*.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *AAAI*, pages 7432–7439.

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

Sahil Chaudhary. 2023. Code alpaca: An instruction-following llama model for code generation. https://github.com/sahil280114/codealpaca.

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

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936.

OpenCompass Contributors. 2023. Opencompass: A universal evaluation platform for foundation models. https://github.com/open-compass/opencompass.

Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2022. Glm: General language model pretraining with autoregressive blank infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 320–335.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2023. Breaking the sequential dependency of llm inference using lookahead decoding.

Jiatao Gu and Xiang Kong. 2021. Fully non-autoregressive neural machine translation: Tricks of the trade. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 120–133.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. In *International Conference on Learning Representations*.

hiyouga. 2023. Llama factory. https://github.com/hiyouga/LLaMA-Factory.

Fei Huang, Hao Zhou, Yang Liu, Hang Li, and Minlie Huang. 2022. Directed acyclic transformer for non-autoregressive machine translation. In *International Conference on Machine Learning*, pages 9410–9428. PMLR.

HuggingFace. 2023. Large language model text generation inference. https://github.com/huggingface/text-generation-inference.

Sehoon Kim, Karttikeya Mangalam, Suhong Moon, Jitendra Malik, Michael W Mahoney, Amir Gholami, and Kurt Keutzer. 2023. Speculative decoding with big little decoder. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. 2017. Race: Large-scale reading comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 785–794.

LAION-AI. 2023. Open-assistant. https://github.com/LAION-AI/Open-Assistant.

Ariel N. Lee, Cole J. Hunter, and Nataniel Ruiz. 2023. Platypus: Quick, cheap, and powerful refinement of llms. *arXiv preprint arXiv:2308.07317*.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR.

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

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

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

Alessandro Palla. 2023. chatbot instruction prompts. https://huggingface.co/datasets/alespalla/chatbot_instruction_prompts.

Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. 2023. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277*.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506.

Benjamin Frederick Spector and Christopher Re. 2023. Accelerating llm inference with staged speculative decoding. In *Workshop on Efficient Systems for Foundation Models@ ICML2023*.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. *Advances in Neural Information Processing Systems*, 31.

Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, Felix Yu, Michael Riley, and Sanjiv Kumar. 2023. Spectr: Fast speculative decoding via optimal transport. In *Workshop on Efficient Systems for Foundation Models@ ICML2023*.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4149–4158.

Paul Tardy. 2023. Rouge. https://github.com/pltrdy/rouge.

InternLM Team. 2023. Internlm: A multilingual language model with progressively enhanced capabilities. https://github.com/InternLM/InternLM.

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

Chunqi Wang, Ji Zhang, and Haiqing Chen. 2018. Semi-autoregressive neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 479–488.

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

Heming Xia, Tao Ge, Furu Wei, and Zhifang Sui. 2022. Lossless speedup of autoregressive translation with generalized aggressive decoding. *arXiv preprint arXiv:2203.16487*.

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

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. 2023. Draft & verify: Lossless large language model acceleration via self-speculative decoding. *arXiv preprint arXiv:2309.08168*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023a. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*.

Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Zihan Wang, Lei Shen, Andi Wang, Yang Li, Teng Su, Zhilin Yang, and Jie Tang. 2023b. Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. 2023. Agieval: A human-centric benchmark for evaluating foundation models.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. 2023. Lima: Less is more for alignment. *arXiv preprint arXiv:2305.11206*.

A Appendix

A.1 Training Details

We conduct all our experiments on a cluster of 4 servers, where each server is equipped with eight A800 (80G) GPUs. We adopt distinct training strategies based on the size of the models being trained. For models with fewer than 14 billion parameters, we allocate our experiments to a single server and employ the ZeRO-2 (Rasley et al., 2020) optimization for distributed training. Conversely, for models that exceed the 14 billion parameter mark, we expand our setup to utilize all four servers and implement the ZeRO-3 optimization to effectively handle the increased computational demands. We adopt LLaMA Factory (hiyouga, 2023) to finetune the LLMs. The specific hyperparameters utilized for the SAR-SFT are documented and can be referenced in Table 2.

Hyperparameters	Value
max source tokens	2048
max target tokens	2048
learning rate	5e-5
scheduler	cosine
Adam β_1	0.9
Adam β_2	0.999
epoch	2
per device batch size	4
gradient clip	1.0

Table 2: Hyper-parameters and training configurations of SAR-SFT.

Table 3 shows the statistics of SFT datasets used to finetuned the models. Note that all the dataset are publicly available. The fine-tuning duration for LLMs can vary significantly based on the size of the model and the computational resources available. For the LLaMA-2-7B model, the fine-tuning process typically takes about 6 hours on a server equipped with eight A800 (80GB) GPUs. For the largest variant, the LLaMA-2-70B, the SAR-SFT requires roughly 18 hours to complete using 4 servers, each equipped with 8 A800 (80GB) GPUs (totalling 32 GPUs).

A.2 Evaluation Details

We performed our inference experiments on a server equipepd with eight A800 (80GB) GPUs. For models with fewer than 14 billion parameters, inference is conducted using a single GPU. For larger models, those with parameters exceeding 14B, we employ multiple GPUs and leverage tensor parallelism to manage the increased computational load effectively. During the inference process, we configure our setup with a batch size of one to ensure precise measurement of inference latency on a per-instance basis.

For generation tasks, we tailored specific prompt templates to guide the model's output. When working with the XSum dataset, we used the following prompt template:"Document: {TEXT}\n Based on the previous text, provide a brief single summary". Similarly, for the HumanEval-X dataset, which is designed for code generation, we employed the prompt template as follows: "Complete the following python code. Do not give any explanation or testing examples, just complete the code.\n {TEXT}". For CIP and MT-Bench, we do not use any prompt template.

To mesure the performance of LLMs on XSum and HumanEval-X, we compute the ROUGE-L and Pass@10, respectively. The ROUGE-L is calculated using python package rouge (Tardy, 2023) and the pass@10 is computed using official evaluation script (Zheng et al., 2023b).

The inference speedup for each task within the MT-Bench benchmark under greedy sampling setting across various models are shown in Table 4.

A.3 Random Sampling

To rigorously evaluate model performance on the XSum and HumanEval-X datasets with random sampling enabled [2], we conducted ten runs of the evaluation process to counteract the influence of randomness. The mean and variance of these runs are reported in Table 5. Under random sampling setting, the output of SPACE and baseline are expected to follow the same distribution, as have been proved in previous work (Chen et al., 2023; Leviathan et al., 2023). The data presented in Table 5 supports this statement, showing that the performance metrics for SPACE and the baseline are similar on both XSum and HumanEval-X. This consistency across multiple evaluations confirms the distributional alignment between SPACE and the baseline model under the random sampling setting.

[2]When using random sampling, we set top-p=0.95 and top-k=10

A.4 SAR-SFT versus SFT

To further demonstrate that SAR-SFT does not impede the model's performance, we compared the performance of LLaMA-2 (with model sizes of 7B, 13B, and 70B parameters) trained with both SAR-SFT and traditional SFT. The comparison spanned a suite of widely used benchmarks, which we have categorized into the following four groups:

- **Academic**. We report the average accuracy of the model on the MMLU (Hendrycks et al., 2020) and AGIEval (Zhong et al., 2023) benchmarks.
- **Knowledge**. We evaluate the model on CommonSenseQA (Talmor et al., 2019) and BoolQ (Clark et al., 2019), reporting their average results.
- **Reasoning**. We assess the 5-shot performance on PIQA (Bisk et al., 2020), RTE (Bentivogli et al., 2009) and HellaSwag (Zellers et al., 2019), reporting their mean performance.
- **Understanding**. We report the average result on RACE (Lai et al., 2017) and SQuAD2.0 (Rajpurkar et al., 2018).

The evaluations were conducted using OpenCompass (Contributors, 2023), an opensource platform designed for large language model evaluation. Comparative performance results are detailed in Table 6. Upon examination of the results, we note small discrepancies between the models fine-tuned with the two distinct training schemes across different tasks.

Dataset	**Language**	**Sample Numbers**	**Average Input Tokens**	**Average Output Tokens**
Alpaca-GPT4-zh (Peng et al., 2023)	zh	48,818	30.9	292.5
Alpaca-GPT4-en (Peng et al., 2023)	en	52,002	21.6	162.6
LIMA (Zhou et al., 2023)	en	1,029	74.2	639.1
Oaast-SFT (LAION-AI, 2023)	multi	20,202	198.8	234.8
CodeAlpaca (Chaudhary, 2023)	en	20,022	28.8	68.6
OpenPlatypus (Lee et al., 2023)	en	24,926	159.6	225.3

Table 3: Statistics of SFT datasets used to finetuned the models. The average input tokens and output tokens are calculated using LLaMA-2-7B tokenizer.

Model	**Code**	**Extrac-tion**	**Human-ities**	**Math**	**Reason-ing**	**Role-play**	**Stem**	**Writ-ing**	**Over-all**
ChatGLM-3-6B	2.83	3.35	1.91	2.54	1.87	1.98	2.03	2.43	2.32
LLaMA-2-7B	2.14	3.12	1.96	2.61	1.83	1.89	1.65	2.25	2.19
LLaMA-2-13B	2.89	3.66	2.20	2.99	2.29	2.03	2.27	2.68	2.53
Qwen-14B	2.88	3.76	2.18	2.85	2.04	1.86	2.05	2.55	2.43
InternLM-20B	2.50	3.28	2.05	3.55	1.89	2.00	2.02	2.05	2.36
Falcon-40B	2.02	2.90	1.72	2.22	1.69	1.53	1.69	1.76	2.17
LLaMA-2-70B	2.61	3.70	1.97	3.03	2.50	1.73	1.84	2.09	2.26

Table 4: The experimental results on MT-Bench under greedy sampling setting. We show the inference speedup for each task in MT-Bench.

Model	**XSum**			**HumanEval-X**		
	ROUGE-L	Avg. Tokens	Speedup	Pass@10	Avg. Tokens	Speedup
ChatGLM-3-6B	14.8 ± 0.2 (14.0 ± 0.4)	1.95 ± 0.01	1.47 ± 0.01	23.2 (22.8)	3.16 ± 0.04	2.09 ± 0.08
LLaMA-2-7B	15.1 ± 0.2 (15.3 ± 0.1)	2.14 ± 0.02	1.79 ± 0.04	18.9 (18.3)	3.56 ± 0.05	2.86 ± 0.04
LLaMA-2-13B	15.2 ± 0.2 (15.6 ± 0.2)	2.24 ± 0.01	1.86 ± 0.02	31.7 (32.3)	4.15 ± 0.02	3.81 ± 0.05
Qwen-14B	16.1 ± 0.3 (16.3 ± 0.3)	2.05 ± 0.01	1.91 ± 0.04	32.3 (31.7)	3.09 ± 0.04	2.86 ± 0.04
InternLM-20B	16.3 ± 0.2 (17.0 ± 0.2)	1.99 ± 0.01	1.73 ± 0.01	25.0 (23.7)	3.13 ± 0.03	2.67 ± 0.08
Falcon-40B	16.6 ± 0.2 (15.4 ± 0.3)	2.09 ± 0.04	2.08 ± 0.03	27.4 (28.0)	3.42 ± 0.03	2.88 ± 0.06
LLaMA-2-70B	16.1 ± 0.2 (16.2 ± 0.3)	2.40 ± 0.02	2.25 ± 0.02	36.6 (38.2)	4.15 ± 0.02	3.81 ± 0.05

Table 5: The experimental results on XSum and HumanEval-X using random sampling. We show the mean and variance (over 10 runs) of the average accepted tokens (Avg. Tokens) and inference speedup (Speedup) for each datasets. The number in parentheses shows the corresponding results of the baseline method.

Model	Scheme	Academic	Knowledge	Reasoning	Understanding
LLaMA-2-7B	SAR-SFT	35.4	66.1	62.3	37.2
	SFT	36.0	65.9	64.1	38.6
LLaMA-2-13B	SAR-SFT	40.9	69.4	66.7	55.2
	SFT	40.5	71.4	65.2	57.4
LLaMA-2-70B	SAR-SFT	50.6	76.7	68.4	64.7
	SFT	51.7	77.2	68.0	66.7

Table 6: Performance comparison of LLaMA-2 (7B, 13B, 70B) with different training schemes.