Make Some Noise: Unlocking Language Model Parallel Inference Capability through Noisy Training

Anonymous ACL submission

Abstract

Existing speculative decoding methods typically require additional model structure and training processes to assist the model for draft token generation. This makes the migration of acceleration methods to the new model more costly and more demanding on device memory. To address this problem, we propose the **Make Some Noise** (MSN) training framework as a replacement for the supervised fine-tuning stage of the large language model. The training method simply introduces some noise at the input for the model to learn the denoising task. It significantly enhances the parallel decoding capability of the model without affecting the original task capability. In addition, we propose a tree-based retrieval-augmented Jacobi (TR-Jacobi) decoding strategy to further improve the inference speed of MSN models. Experiments in both the general and code domains have shown that MSN can improve inference speed by 2.3-2.7x times without compromising model performance. The MSN model also achieves comparable acceleration ratios to the SOTA model with additional model structure on Spec-Bench.

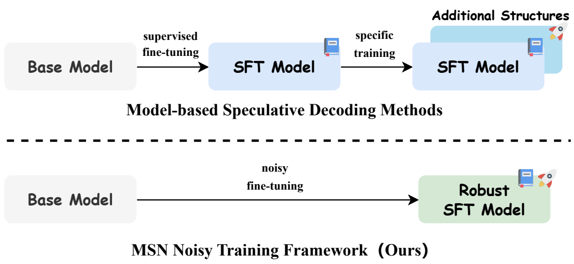

Figure 1: An illustration of the differences between the proposed MSN framework and existing model-based speculative decoding methods. The book icon represents task-specific capabilities and the rocket icon represents parallel decoding capabilities.

1 Introduction

Large language models (LLMs) represented by GPT-4 (OpenAI et al., 2024) and LLaMA (Touvron et al., 2023) have made great breakthroughs to artificial intelligence (Kocoń et al., 2023). However, LLMs suffer from high inference latency due to the autoregressive (AR) decoding paradigm, which constrains the model to generate only one token per decoding step. It significantly limits the applications of LLMs when needs lengthy response.

To address the bottleneck introduced by AR, speculative decoding (Leviathan et al., 2023; Chen et al., 2023) is proposed to get more than one token in one decoding step. It first guesses multi-step draft tokens and then verifies them simultaneously in one model forward. Once any draft token is accepted, it can effectively speedup the inference process. Chen et al. (2023) employ a relatively small LLM to generate multi-step draft tokens and verify them in parallel on the target LLM. Medusa (Cai et al., 2024) extends and train multiple language model heads for existing models to predict later draft tokes. It achieves considerable inference speedup through efficient validation using tree attentions. BiTA (Lin et al., 2024a) takes full advantage of the capabilities of LLM itself through a parameter-efficient design that allows the model to generate daft tokens based on trainable special tokens. Kou et al. (2024) propose a post-training method based on constructed Jacobi trajectories that can accelerate the model's own Jacobi decoding capabilities.

Although the above methods improve the inference efficiency of the model to a certain extent, there are still some problems to be solved as shown in Figure 1. (1) **Additional Structures**. Most current speculative decoding methods rely heavily on additional model structures to accomplish draft token prediction (e.g., separate models, language model heads, trainable prompts, etc.). In the case of Medusa, for example, it adds 1.6B parameters (5 additional medusa heads) to the 7B target model, which will undoubtedly increase the memory requirements for model inference. (2) **Separate Post-Training**. Existing model-based speculative decoding methods are trained after LLMs' supervised fine-tuning (SFT) stage to obtain acceleration capability. This process usually requires complex model setups or time-consuming data construction, and some methods even lose part of the model's original task capabilities. Separate training of task and acceleration capabilities leads to an overly complex approach which is not easy to deploy.

To address the above problem, we propose a noisy training framework [1] **M**ake **S**ome **N**oise (MSN) as a replacement for SFT, which enables the model to acquire both task-relevant capability as well as acceleration capability at the same stage without the need for additional structures and training stages. Specifically, we consider the process of Jacobi decoding (Santilli et al., 2023) as a denoising process, and improve the denoising ability of the model by including a causal language model denoising task in the SFT stage. Since the SFT stage is almost a necessary aspect of LLM applications, our proposed approach can be interpreted as a **free lunch** to the parallel inference capability of LLMs. In the inference phase, we use Jacobi decoding to achieve inference acceleration through repeated iterations of random noise tokens as well as verification. Besides, in order to alleviate the cold-start problem of Jacobi decoding and mitigate the effect of random initial noise, we also propose the tree-based retrieval-augmented Jacobi (TR-Jacobi) decoding method, which can effectively improve the speedup ratio.

We have conducted detailed experiments in the general and code domains. The results show that the MSN training framework can significantly improve the denoising ability of the model without affecting the performance of the original SFT model, which in turn achieves a 2.3-2.7x inference acceleration effect. In addition, we performed a detailed evaluation on Specbench, which is specifically designed for speculative decoding. As a speculative decoding method without additional structure and training, the acceleration ratio of the MSN model under TR-Jacobi decoding strategy significantly outperforms other additional-structure-free methods and possesses comparable speedup ratios to the SOTA model with additional model structure and training.

[1] https://github.com/XXX

Our main contributions can be summarised as follows:

- We propose a new training framework Make Some Noise (MSN) as an alternative to SFT, which can unlock the parallel decoding capability of the model through the denoising task.
- We propose a tree-based retrieval-augmented decoding method that effectively improves the inference speed of MSN models under memory bottlenecks.
- Experiments show that MSN training enables the model to have a comparable acceleration ratio to the SOTA method without significant loss of task performance.

2 Related Work

2.1 Jacobi Decoding

Jacobi decoding (Santilli et al., 2023) treats greedy decoding of generative tasks as solving equations:

$$\begin{cases} y_1 & = \arg\max P_\theta(y_1|x) \\ y_2 & = \arg\max P_\theta(y_1|y_1, x) \\ & \vdots \\ y_m & = \arg\max P_\theta(y_m|y_{1:m-1}, x) \end{cases} \tag{1}$$

Auto-regressive decoding solves the equations from first to last based on the given input x, progressively replacing the resolved variables. In contrast, Jacobi decoding relies on Jacobi and Gauss-Seidel (GS) fixed-point iteration methods (Ortega and Rheinboldt, 2000) to solve Equation 1 in parallel. Specifically, it passes an initialisation sequence of length m into the model for iterative generation until the sequence converges to a fixed point. Jacobi decoding expects to solve the equation in less than m iterations, but in fact existing models perform poorly under this decoding strategy due to the lack of denoising capability. Kou et al. (2024) greatly improve the efficiency of Jacobi decoding by constructing the trajectory data during Jacobi decoding and performing consistency training.

2.2 Speculative Decoding

Speculative decoding can effectively increase the decoding speed without changing the output quality by guessing and verifying the output of the auto-regressive language model in parallel. Current mainstream work has focused on investigating how

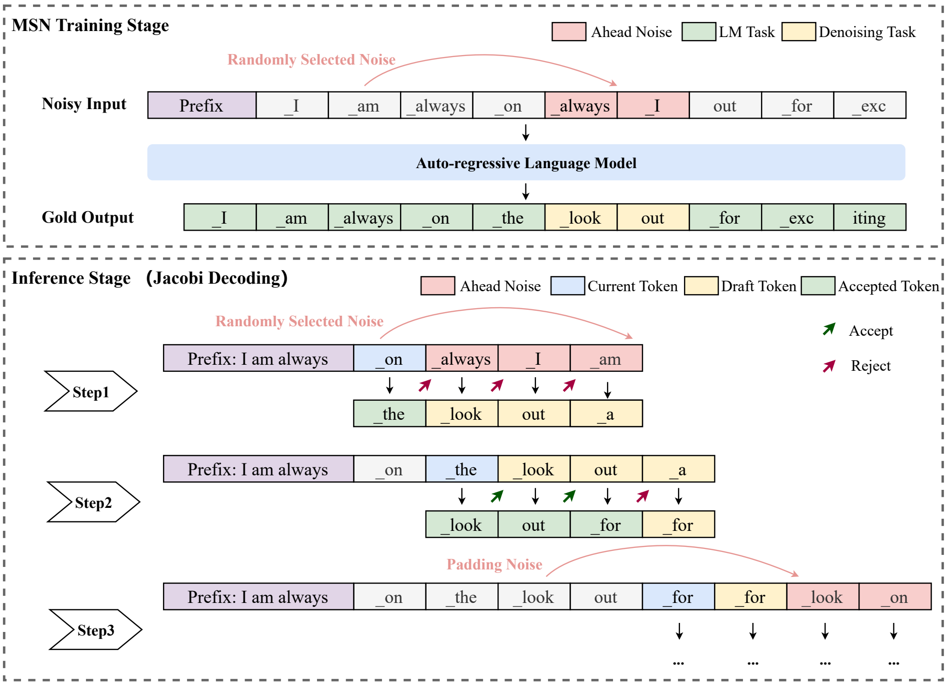

Figure 2: Illustrations of the Make Some Noisy training framework and Jacobi decoding strategy. The training phase in the figure uses a noise segment of length 2, and the inference phase is shown as an example when the length of the noise segment is set to 3.

to complete draft token generation efficiently. Stern et al. (2018) complete the prediction of draft tokens with additional model structures. Chen et al. (2023) generate reliable draft tokens by a external small model. Cai et al. (2024) train multiple heads for the LLM model for predicting draft tokens based on the previous work. Li et al. (2024) make full use of the information in the hidden layer to accomplish high-quality predictions of draft models with a separate decoder layer. Lin et al. (2024a) enable the model to predict draft tokens by training prefix tokens.

In addition, there are speculative decoding methods that do not require training. Fu et al. (2024) performs more efficient verification by collecting n-gram segments generated during Jacobi decoding as draft tokens. Saxena (2023) achieves acceleration in specific domains simply by retrieving draft tokens from the ahead prompt. He et al. (2023) enable plug-in draft token generation by retrieving a constructed knowledge database.

In order to further improve the verification efficiency of draft token, Miao et al. (2023) propose to verify multiple paths as a token tree at a time by designing an attention mask matrix. Nowadays, token tree verification has become a widely used technique to improve the verification efficiency of speculative decoding.

3 Method

3.1 Overall

Our core idea is to consider parallel decoding as a kind of text generation under noise, similar to the Jacobi decoding. This requires the model to have the ability to generate the corresponding correct token despite the noisy token, which is not possible with the current teacher-forcing training (Bachmann and Nagarajan, 2024).

Inspired by related work addressing exposure bias (Bengio et al., 2015; Zhang et al., 2019), we chose to enhance the denoising ability of the model by adding some token-level noise to the input sequence in the SFT stage of LLMs. As shown in Figure 2, we incorporate a causal language model denoising task in the training phase to ensure that the model has the robust generation capability. During the inference phase, we use random noise spliced at the end of the sequence, and keep generating and verifying draft tokens by iterative denoising, consistent with the Jacobi decoding process.

To guarantee that the denoising ability is improved without affecting the acquisition of task capabilities, we construct the method in terms of the content and location of the noise segment(Section 3.2). In addition, to further enhance the validation efficiency of the model, we propose a tree-based retrieval-augmented Jacobi (TR-Jacobi) decoding strategy (Section 3.3).

3.2 Noisy Training Framework

Teacher-forcing has been widely adopted as an efficient training method by the dominant generative models. It trains the model with the label at moment t as the input at moment $t+1$, which can accelerate the model convergence. For the sequence $X = x_0 x_1 ... x_n$, the loss function of a traditional auto-regressive model can be formulated as:

$$Loss_{AR} = \sum_{i=0}^{n} -\log P(X_i | X_{<i}; \theta) \quad (2)$$

where θ is the set of parameters of the language model and $X_{<i}$ represents the sub-sequence $x_0 x_1 ... x_{i-1}$. The model is trained to generate results based on the correct labels, therefore each generation step requires the results generated in the previous step.

In order to equip the model with denoising capability, we introduce causal noise token in the training phase. As shown in Figure 2, we insert some noise tokens at the input to break the restriction that teacher-forcing always takes golden labels as input. To minimise the impact of noise on training, we only replace one short segment with noise tokens in each sample. The noise sample can be expressed as $\hat{X} = x_0 x_1 ... \hat{x}_i ... \hat{x}_j ... x_n$ where $\hat{x}_i ... \hat{x}_j$ represents the noise segment. The loss function of the noisy training method can be formulated as:

$$Loss_{MSN} = \sum_{i=0}^{n} -\log P(X_i | \hat{X}_{<i}; \theta) \quad (3)$$

where X_i represents the token of golden labels and $\hat{X}_{<i}$ represents the sub-sequence with noise tokens. It should be noted that even though the input contains partially noise tokens, the target of the model to learn is still the correct labels. Such training with noise can unlock the parallel decoding capability of the model to some extent. To further reduce the impact of noise on the SFT task, we investigate the content of the noise and the location of the noise.

The Content of the Noise Segment. The main motivation for noisy training is to equip the model with the ability to generate correct tokens despite noisy inputs, which is achieved through the loss of the noise segments. However, the causal attention mask of the LLMs leads to the possibility that the noise tokens may have an impact on the later auto-regressive training objectives. To minimise the impact, we chose the ahead noise as the main content of the segments. Specifically, we randomly sample the ahead tokens as the current noise token, which can be formulated as:

$$\hat{x}_i = random\_sample(X_{<i}) \quad (4)$$

where $X_{<i}$ represents for the sub-sequence ahead of x_i. Compared to random noise, ahead noise has less impact on subsequent tokens. In addition, denoising the ahead noise tokens is more challenging since they are more relevant to the context.

The Location of Noise Segment. Inspired by Lin et al. (2024b), we have tried two noise location selection methods, random selection and PPL-based selection. Experiments (see the Appendix A for details) have found that neither method has a significant impact on the model task performance and the speedup ratios are similar. We speculate that our noise segments (less than 10) may be relatively short on SFT datasets with an average length of 600 or more, and do not have an impact on the training of the model itself. We therefore choose the simpler random replacement noise method.

In practice, at each step of training, we only replace one fixed-length random segment with ahead noise for the response of each sample.

3.3 TR-Jacobi Decoding

Tree-based Jacobi Decoding. As discussed in Section 2.2, using token tree verification has become a common method of verification in speculative decoding. In this paper, we also want to improve the efficiency of Jacobi decoding by constructing multiple candidate sequences. Like Medusa (Cai et al., 2024), we heuristically chose a sparse tree as our tree-attention template (see the Appendix B for details). At the beginning of the generation, we initialise all the nodes of the tree using ahead noise to start the tree-based Jacobi decoding. As shown in Figure 3, for each forward process, each path performs an ordinary Jacobi decoding process via tree attention. We then choose the longest accept-length path and continue to fill the validation tree nodes for next round based on the path's subsequent predictions. It is important to note that we use the ahead noise tokens to populate the remaining positions in the validation tree, just like regular Jacobi decoding.

Retrieval-Augmented Jacob Decoding. In addition, for methods that design draft token predictions on the input side of the model (e.g., Jacobi, BiTA, etc.), cold-start is also a key issue that needs to be addressed. When all draft tokens of this input are accepted, the model will have no way to get new draft tokens in this round. Existing methods mitigate this problem by subsequently splicing more tokens, but incur additional inference costs. To avoid starting validation from completely random noise in this case, we consider combining retrieval-based draft token and model-based draft token generation.

Specifically, we set a retrieval path in the token tree to hold the candidate tokens obtained by retrieving the previous tokens. For retrieval, we use a simple and efficient method called prompt lookahead decoding (Saxena, 2023) to obtain draft tokens with the same beginning directly from the current ahead tokens for verification, which significantly accelerates inference on tasks such as summarization. The analysed experiments in Section 5.3 demonstrate that incorporating retrieved information is effective in improving the model's acceleration ratio in specific domains. Also, Jacobi decoding can alleviate the inherent problems of retrieval methods in domains such as translation.

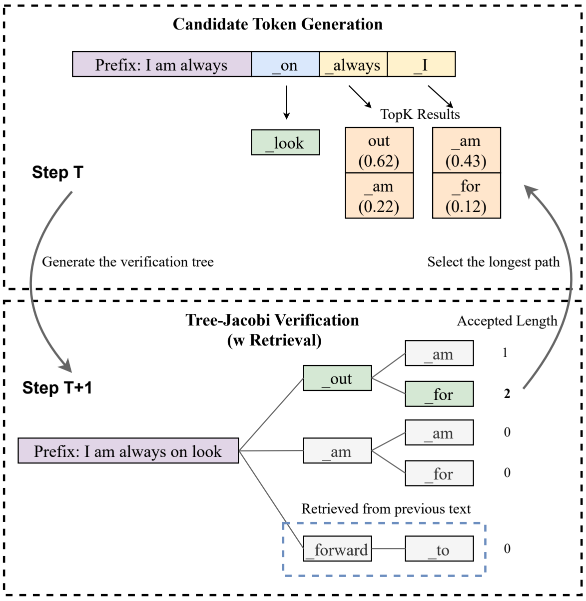

Figure 3: The main flowchart of TR-Jacobi decoding. It should be noted that candidate generation and tree verification are performed **in the same step**. For clarity, we choose candidate generation at moment T and tree verification at moment T+1 for analysis in the figure.

4 Experiments

4.1 Experimental Setup

Datasets. To verify that our proposed Make Some Noise (MSN) SFT training can bring inference acceleration without compromising model performance, we have constructed SFT datasets in the general and code domains, respectively. For the general domain, we follow Lin et al.'s (2024a) setup to construct a training dataset containing 190k samples from LIMA (Zhou et al., 2024), Alpaca-GPT4 (Peng et al., 2023), CodeAlpaca (Chaudhary, 2023), OpenPlatypus (Lee et al., 2023) and CIP (Palla, 2023). Note that we only use 100k samples from CIP. For the code domain, we adopt a total of 185k samples from Magicoder-OSS (Wei et al., 2023) and Evol-CodeAlpaca (Luo et al., 2023) as the training dataset, which are widely used in the program synthesis task.

Training Settings. To evaluate the proposed method comprehensively, we select LLama3-8B-Base (Touvron et al., 2023) and DeepseekCoder-6.7b-Base (Guo et al., 2024) as the foundation models for the general and code domains, respectively. The training settings for MSN are aligned with the baseline (SFT), maintaining a sequence length of 2048 tokens, a batch size of 512, and a training epoch of 4. Full-parameter fine-tuning is performed on two servers, each equipped with 8 A100-80GB GPUs, utilizing bf16 precision. We determine that a noise segment length of 4 is optimal for dynamic noise replacement for each sample.

Evaluation Settings. In this paper, we conduct experiments on the task performance and acceleration performance of MSN, respectively. For task performance, we use the MT-bench (Zheng et al., 2024) in the general domain, the HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) benchmarks in the code domain for evaluation. Evalplus (Liu et al., 2023) , which provides additional test cases for problems in HumanEval and MBPP, is also included. For acceleration performance, we performed a speedup evaluation of the proposed parallel decoding methods on Spec-Bench (Xia et al., 2024). This benchmark contains data from multiple domains and provides a fair comparison with existing acceleration methods. Following previous work, all speed related experiments are done on a single A100-80G device with the batch size as 1. For our MSN model, the draft token length during inference is consistent with the noise segment length during training, which is 4.

Methods	Metric(+)	Speed (tokens/s)	Speedup
LLaMA3-8B-Base (General)			
SFT		38.00	1.00×
+Jacobi	**6.13**	39.38	1.01×
+TR-Jacobi		72.04	1.90×
MSN (Ours)		44.69	1.00×
+Jacobi	6.12	72.53	1.62×$_{\uparrow 60}$
+TR-Jacobi		99.32	2.28×$_{\uparrow 20}$
DeepseekCoder-6.7B-Base (Code)			
SFT		48.47	1.00×
+Jacobi	76.6 (68.6)	48.59	1.00×
+TR-Jacobi		90.81	2.08×
MSN (Ours)		48.03	1.00×
+Jacobi	**77.0** (**68.7**)	89.73	1.97×$_{\uparrow 97}$
+TR-Jacobi		128.50	2.68×$_{\uparrow 29}$

Table 1: Results of task performance experiments in general and code domains. The general domain metric uses scores from MT-bench and the code domain uses pass@1 under greedy decoding. For the code domain, we choose the average of HumanEval and MBPP as a composite metric. '(+)': Results after executing additional tests from evalplus. '↑': Percentage improvement over models without MSN.

4.2 Comparison with SFT

Baselines. We first validate the impact of the proposed MSN training framework on the performance of the model tasks in the general and code domains. The standard supervised fine-tuning (SFT) is chosen as the baseline method for comparison. Specifically, we perform domain-specific SFT and noise training based on the same base model and compare the performance of both on downstream tasks.

Results. The metric in Table 1 represents the task performance of each model. There is no significant performance loss of the model trained by MSN on the downstream task compared to SFT. Futhermore, The MSN model even delivers a slight performance boost in both domain. The enhancement in the code domain is particularly noteworthy, given that evaluating generated programs is more rigorous than evaluating conversation. Programs must be correctly formatted and pass all test cases to be deemed successful. It indicates that MSN does not hurt the model to acquire capabilities during the SFT phase. Our analysis suggests that this gain comes from the fact that noise mitigates the negative effects of teacher forcing training on the model to some extent. The causal denoising task forces the model to focus on more distant tokens when predicting the current location token because the current input is noisy.

In addition to this, we briefly test the acceleration effect of the MSN method on the Jacobi-like decoding strategy. We can see that targeted training on the denoising ability of the model significantly improves the acceleration ratio of Jacobi decoding in different domains. Our proposed TR-Jacobi further improves the acceleration ratio by verifying multiple paths simultaneously.

4.3 Comparison with Other Speculative Decoding Methods

Baselines. To further compare MSN with existing speculative decoding methods, we conducted an evaluation on Spec-Bench (Xia et al., 2024). We choose both speculative methods that include no additional structures (Jacobi, LookAhead, PLD) and those that require additional structures (Medusa2, EAGLE) for comparison. EAGLE and Medusa2 are post-trained on Vicuna-7b-v1.3 (Chiang et al., 2023), which is already a post-SFT model. Since our proposed MSN is performed in the SFT stage, we need to perform MSN SFT on a base model. Therefore, we conduct MSN on LLaMA3-8B-Base and perform acceleration evaluations on two different foundation models for a rough comparison of speedup ratios based on different auto-regressive (AR) throughputs.

Results. The overall acceleration experiment results are shown in Table 2. After specific training on denoising capabilities, the MSN model improves the speedup ratio on all Jacobi-like decoding strategies. For LookAhead, the denoising ability may produce incoherent n-grams, which can lead to a relatively low improvement. For both Jacobi decoding and TR-Jacobi decoding acceleration ratios, noisy training brings significant improvements. TR-Jacobi has a fine blend of retrieved and generated draft tokens with respectable average receive lengths in all domains.

The speedup ratio of the MSN model under TR-Jacobi decoding is competitive with other methods. As a method with no additional training stages and no additional model structure, the proposed acceleration method is also comparable to the models with additional structures. It is fair to say that MSN is a lightweight and efficient way to achieve inference speedup comparable to existing SOTA models while improving model robustness.

Methods	AS	MT-B	Trans	Sum	QA	Math	RAG	#MAT	#Speed (tokens/s)	Overall
					Vicuna-7B-v1.3					
AR	✗	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00	49.64	1.00×
PLD	✗	1.60×	1.03×	2.58×	1.15×	1.72×	2.15×	1.85	84.23	1.69×
Medusa2	1.6B	2.54×	2.01×	2.22×	2.00×	2.59×	2.09×	3.12	111.49	**2.25**×
EAGLE	0.3B	2.59×	1.91×	2.25×	2.07×	2.61×	2.01×	3.58	111.58	**2.25**×
LookAhead	✗	1.44×	1.14×	1.31×	1.26×	1.57×	1.21×	1.65	65.80	1.32×
Jacobi	✗	0.95×	0.92×	0.94×	0.94×	0.98×	0.94×	1.05	47.06	0.95×
TR-Jacobi	✗	1.69×	1.31×	2.10×	1.28×	1.74×	1.58×	2.00	80.30	1.62×
					LLaMA3-8b-MSN (**Ours**)					
AR	✗	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00	42.13	1.00×
LookAhead	✗	1.51×	1.36×	1.46×	1.35×	1.65×	1.40×	1.75	61.51	1.46×$_{\uparrow 11}$
Jacobi	✗	1.62×	1.54×	1.75×	1.41×	1.67×	1.48×	1.86	66.68	1.58×$_{\uparrow 66}$
TR-Jacobi	✗	2.22×	2.03×	2.77×	1.85×	2.16×	1.96×	2.94	91.63	**2.17**×$_{\uparrow 34}$

Table 2: Experimental results of acceleration ratios in various areas of Spec-Bench (Multi-turn Conversation, Translation, Summarization, Question Answering, Mathematical Reasoning, Retrieval-aug. Generation). Under the dashed line indicates the Jacobi-like decoding method. 'AS': Additional Structure. '#MAT': Mean Accepted Tokens. '↑': Percentage improvement over models without MSN.

L	HEval(+)	MBPP(+)	Speed (tokens/s)	Speedup
1	**74.4** (**70.1**)	75.9 (64.6)	58.35	1.55×
4	73.2 (68.3)	**76.5** (64.3)	80.47	**2.13**×
8	71.3 (65.9)	76.5 (**65.1**)	80.02	2.12×

Table 3: The effect of the training noise segments length on acceleration and task capability. 'L' represents the length of the noise segment.

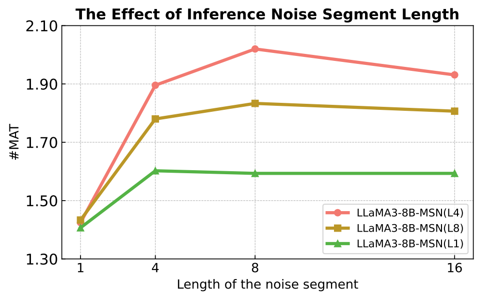

Figure 4: The effect of the inference noise segments length on acceleration with Jacobi decoding. 'MAT': Mean Accepted Token.

5 Discussion

5.1 Effect of Noise Segment Length

The span length includes the length of the noise segment during training and the length of the draft sequence added during inference. The training span length affects the difficulty of the model learning from samples, while the span length during inference impacts both the hit length and the speculative operation latency.

Training Noise Segment length. Traning Noise segment length refers to the number of noise tokens. If the length is too short, the denoising capability of the model may be diminished, resulting in limited acceleration during inference. Conversely, if the length is too long, it significantly increases the difficulty of denoising, affecting the model's understanding of the sample and thereby harming its task performance. To observe the impact of varying training span lengths, we experiment with span lengths of 1, 4, and 8 on Deepseek Coder and the task performance and acceleration are shown in Table 3. It demonstrates that a length of 1 yields high task performance but offers minimal acceleration. A length of 8 provides substantial acceleration but at the cost of significant task performance degradation. A length of 4 achieves the highest acceleration with a lower impact on performance.

Inference Noise Segment Length Inference noise segment length represents the draft token num for Jacobi iteration, which is also the maximum number of times the token can be iteratively denoised. We perform parallel inference experiments with different inference noise segment lengths for models trained with different training noise segment lengths (described above). We find that the model can generalize from a smaller train-

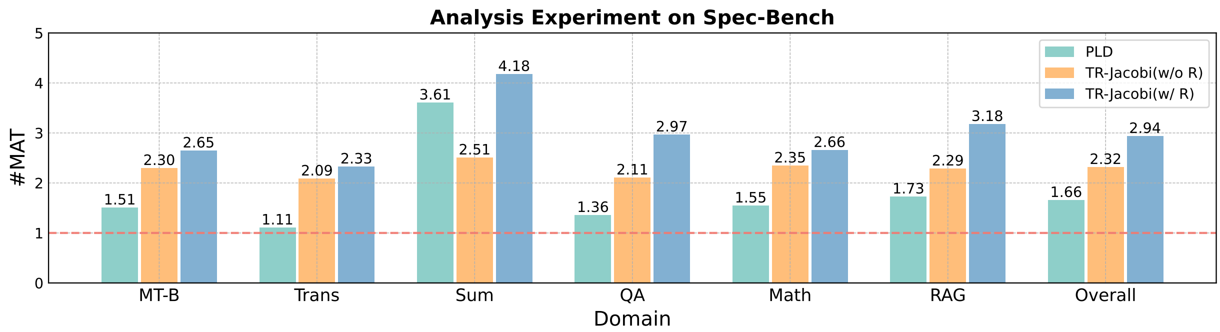

Figure 5: Results of ablation experiments on the retrieval part of TR-Jacobi decoding.

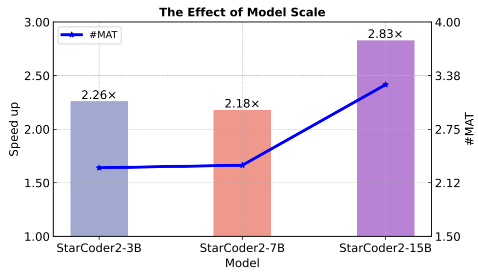

Figure 6: Acceleration experimental results of MSN training for StarCoder2 models of different sizes.

ing noise segment size to a larger inference noise segment size. This suggests that even though we only trained one step to go directly from noise token to gold token, the model is able to generalize to obtain iterative denoising ability. In addition, the training noise length of 8 does not outperform the training noise length of 4, suggesting that length 4 has reached the bottleneck of the model's denoising ability in the SFT stage.

5.2 Effect of Model Scale

To assess the generalisation capability of MSN, experiments are conducted on different sizes of Starcoder2 (Lozhkov et al., 2024), specifically 3B, 7B, and 15B parameters. The training data remains consistent with Section 4.1, and HumanEval with Jacobi decoding is utilised to evaluate the acceleration. The results of the experiment are shown in Figure 6. Overall, MSN demonstrates significant speedup across all model sizes, indicating its broad applicability.

Specifically, when increasing the model size from 3B to 7B, the Mean Accepted Tokens (#MAT) only increases by 0.03, and the speedup ratio slightly decreases. It suggests that a 3B model is sufficient to learn the denoising capability and that the effectiveness of denoising does not significantly change with an increase in parameters from 3B to 7B. The incremental increase in MAT for the 7B model is insufficient to offset the additional computational cost of draft tokens during inference, resulting in a decrease in the speedup ratio. However, when the model size reaches 15B, the denoising capability increases dramatically. The #MAT rises by nearly 1, and the additional computational cost of draft tokens is mitigated by the substantial improvement in hit rate, resulting in a 0.6 increase in the speedup ratio. The outcomes on model scale further exemplify the extensive applicability of our method and demonstrate that larger models have greater potential.

5.3 Effect of Retrieval Paths

In order to further analyse the performance enhancement brought by the retrieval paths to TR-Jacobi decoding, we perform ablation experiments with Llama3 on Mt-Bench. We compare the #MAE for the pure retrieval method PLD, the pure Jacobi method TR-Jacobi w/o R, and TR-Jacobi on each domain. The results of the experiment are shown in Figure 5. Our proposed TR-Jacobi integrates and surpasses pure Jacobi and pure retrieval solutions in terms of acceleration performance in various domains. Retrieval paths mitigate the cold start and instability due to random noise of Jacobi's approach. The Jacobi method can continue to iterate over the retrieval path and can also handle tasks with shorter contexts (e.g., translation).

6 Conclusion

In this paper, we propose an effective training framework Make Some Noise (MSN) to be used as a replacement for the SFT stage. It enhances the denoising ability of the model without affecting the SFT training performance. Combined with our proposed TR-Jacobi decoding strategy, the MSN model is able to achieve 2.3-2.7x speedup in the general and code domains without additional structure and training.

Limitations

Causal denoising, as a more general task, is only used for experiments in the SFT phase in this paper due to limited computational resources. It is a worthy exploration to merge the denoising task with the next token prediction task into the pre-training task. In addition to this, the optimal noise fragment length may be related to the content of the SFT training set (parallel prediction of code text is less difficult, natural language text is more difficult). For a new SFT dataset, confirming the optimal noise segments may require some pre-experiments for searching, which imposes a certain burden on MSN training.

Ethics Statement

The source data for proposed methods come exclusively from publicly available project resources on legitimate websites and do not involve any sensitive information. In addition, all baselines and datasets used in our experiments are also publicly available, and we have acknowledged the corresponding authors by citing their work.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Gregor Bachmann and Vaishnavh Nagarajan. 2024. The pitfalls of next-token prediction. *arXiv preprint arXiv:2403.06963*.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. *Advances in neural information processing systems*, 28.

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

Sahil Chaudhary. 2023. Code alpaca: An instruction-following llama model for code generation. *Code alpaca: An instruction-following llama model for code generation*.

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*.

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. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

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

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

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y Wu, YK Li, et al. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

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

Jan Kocoń, Igor Cichecki, Oliwier Kaszyca, Mateusz Kochanek, Dominika Szydło, Joanna Baran, Julita Bielaniewicz, Marcin Gruza, Arkadiusz Janz, Kamil Kanclerz, et al. 2023. Chatgpt: Jack of all trades, master of none. *Information Fusion*, 99:101861.

Siqi Kou, Lanxiang Hu, Zhezhi He, Zhijie Deng, and Hao Zhang. 2024. Cllms: Consistency large language models. *arXiv preprint arXiv:2403.00835*.

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.

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

Feng Lin, Hanling Yi, Hongbin Li, Yifan Yang, Xiaotian Yu, Guangming Lu, and Rong Xiao. 2024a. Bita: Bi-directional tuning for lossless acceleration in large language models. *arXiv preprint arXiv:2401.12522*.

Zhenghao Lin, Zhibin Gou, Yeyun Gong, Xiao Liu, Yelong Shen, Ruochen Xu, Chen Lin, Yujiu Yang, Jian Jiao, Nan Duan, et al. 2024b. Rho-1: Not all tokens are what you need. *arXiv preprint arXiv:2404.07965*.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh*

Conference on Neural Information Processing Systems.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2024. Starcoder 2 and the stack v2: The next generation. *Preprint*, arXiv:2402.19173.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023. Wizardcoder: Empowering code large language models with evol-instruct.

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*, 1(2):4.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

James M Ortega and Werner C Rheinboldt. 2000. *Iterative solution of nonlinear equations in several variables*. SIAM.

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*.

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

Apoorv Saxena. 2023. Prompt lookup decoding.

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

Chenxi Sun, Hongzhi Zhang, Zijia Lin, Jingyuan Zhang, Fuzheng Zhang, Zhongyuan Wang, Bin Chen, Chengru Song, Di Zhang, Kun Gai, et al. 2024. Decoding at the speed of thought: Harnessing parallel decoding of lexical units for llms. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 4476–4487.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. 2023. Magicoder: Source code is all you need. *arXiv preprint arXiv:2312.02120*.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. 2024. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. *arXiv preprint arXiv:2401.07851*.

Wen Zhang, Yang Feng, Fandong Meng, Di You, and Qun Liu. 2019. Bridging the gap between training and inference for neural machine translation. *arXiv preprint arXiv:1906.02448*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2024. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. 2024. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems*, 36.

A PPL-Based Location Selection

As discussed in Section 3.2, we try to use the PPL-based selection method to select the location of the noise segments. Inspired by Lin et al. (2024b), different tokens contribute differently to the learning of that sample. Therefore, we consider using cross-entropy loss to score the input tokens and select the segment with the lowest loss for noise replacement, which can be formulated as:

$$k = \arg\min \sum_{i=k}^{k+l} -\log P(X_i|X_{<i};\theta) \quad (5)$$

where k represents the start index of the noise segment and l represents the length of the noise segment. Segments with low cross-entropy loss possess both correct prediction and high prediction confidence. Correct predictions indicate that the model has learnt this segment sufficiently and replacement with noise has minimal impact on model performance. High prediction confidence means that the segment is likely to be a commonly used expression (Sun et al., 2024), which is useful for learning acceleration capabilities.

The final results of the experiment are shown in Table 4. Even in code domains with stringent output requirements, ppl-based position selection has no significant speed or performance advantage over random selection. Considering that the ppl-based training method is too complicated and increases the training time to some extent, we subsequently adopt random noise locations.

B Templates for Token Tree

As shown in Figure 7, token tree verification organizes multiple paths into a tree structure, which is verified in parallel by sparse attention masks. With high accuracy of draft token prediction, token tree verification can effectively improve the average acceptance length. However, for Jacobi decoding, since no additional structure is introduced, the correct prediction rate of its draft token is relatively low, and the generation of draft fragments is mainly achieved by iterative decoding. Therefore the enhancement brought by tree verification mainly depends on the topK of the first draft token,

	HumanEval (+)	MBPP (+)	Speed (tokens/s)	Speedup
Baseline	77.4 (72.6)	75.7 (64.6)	44.01	1.00×
Random	76.8 (72.0)	75.4 (65.1)	99.96	2.11×
PPL-Based	77.4 (70.7)	76.5 (66.7)	101.18	2.13×

Table 4: The comparison between the randomly selected noise segment and the lowest loss noise segment.

and experiments show that TR-Jacobi decoding is not sensitive to the structure of the verification tree. In this paper, we use the same heuristic tree structure as vicuna-7b in medusa (Cai et al., 2024), containing 63 nodes. In particular, we also add a retrieval path of length 5 to store the retrieved draft tokens.

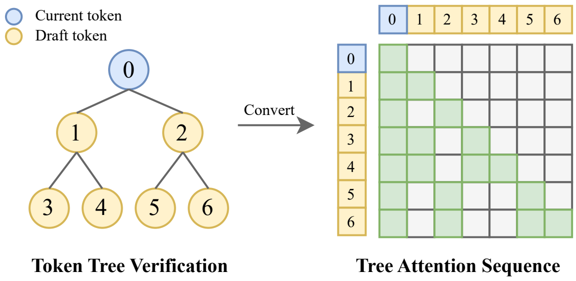

Figure 7: Illustration of token tree verification. The model achieves simultaneous verification of multiple candidate paths through a specially constructed sparse attention matrix.