PEARL: PARALLEL SPECULATIVE DECODING WITH ADAPTIVE DRAFT LENGTH

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding (SD), where an extra draft model is employed to provide multiple *draft* tokens first and then the original target model verifies these tokens in parallel, has shown great power for LLM inference acceleration. However, existing SD methods suffer from the mutual waiting problem, i.e., the target model gets stuck when the draft model is *guessing* tokens, and vice versa. This problem is directly incurred by the asynchronous execution of the draft model and the target model, and is exacerbated due to the fixed draft length in speculative decoding. To address these challenges, we propose a conceptually simple, flexible, and general framework to boost speculative decoding, namely **P**arallel sp**E**culative decoding with **A**daptive d**R**aft **L**ength (PEARL). Specifically, PEARL proposes *pre-verify* to verify the first draft token in advance during the drafting phase, and *post-verify* to generate more draft tokens during the verification phase. PEARL parallels the drafting phase and the verification phase via applying the two strategies, and achieves adaptive draft length for different scenarios, which effectively alleviates the mutual waiting problem. Moreover, we theoretically demonstrate that the mean accepted tokens of PEARL is more than existing *draft-then-verify* works. Experiments on various text generation benchmarks demonstrate the effectiveness of our PEARL, leading to a superior speedup performance up to **4.43**$\times$ and **1.50**$\times$, compared to auto-regressive decoding and vanilla speculative decoding, respectively.

1 INTRODUCTION

Large language models (LLMs) such as GPT-4, LlaMA, and Claude 3 (Achiam et al., 2023; Team, 2024; Bommasani et al., 2021; Touvron et al., 2023) have dominated natural language understanding and generation (Khurana et al., 2023) over a wide range of applications. However, the substantial inference latency of these LLMs has emerged as a significant obstacle bounding their broader application in scenarios with restricted computational resources. This latency primarily originates from the auto-regressive token-by-token decoding process wherein decoding K tokens requires K serial runs of LLMs, incurring exacerbated latency with both the length of generated tokens and the model scale.

To address this challenge, extensive research efforts have been devoted to accelerating LLM inference. Given that inference from large models is often constrained more by memory bandwidth and communication than by arithmetic operations Leviathan et al. (2023), one innovative inference paradigm, Speculative Decoding (SD), has emerged as a new trend and shown superior performance by effectively enabling better GPU utilizations. As shown in the upper part of Figure. 1, the key idea of SD algorithm is to employ an extra small model (referred as the *draft model*) to firstly generate γ draft tokens for the original large model (referred as the *target model*), and then the target model verifies these draft tokens in parallel within a single forward. Here, γ is a fixed hyperparameter **window size**. **Draft length** is the number of tokens generated by the draft model in a continuous execution. Therefore, the draft length is set to γ in SD. Following-up works effectively extend this framework by either removing the necessity of the draft model (Cai et al., 2024; Fu et al., 2024; Zhang et al., 2023) or identifying a compact draft model with high distribution alignment (Zhou et al., 2023; Zhao et al., 2024; Miao et al., 2023). Extensive experiments demonstrate that this *draft-then-verify* framework effectively enhances the concurrency of the target model, thereby significantly accelerating the inference process.

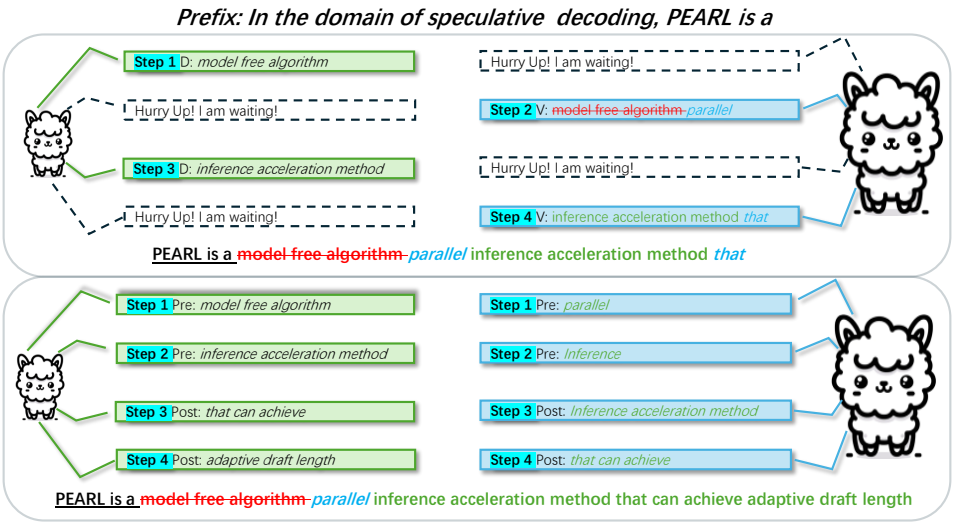


Figure 1: An overview of speculative decoding (the upper part) and our PEARL (the lower part). SD employs a draft model to provide multiple drafts and then the target model verifies the drafts in parallel. However, SD suffers from the mutual waiting problem, i.e., the target model gets stuck when the draft model is *guessing* tokens, and vice versa (the dashed dialogue box). PEARL parallels the drafting and verification process to alleviate the mutual waiting problem. Moreover, PEARL can leverage adaptive draft length to generate more tokens within the same amount of time to further mitigate the mutual waiting problem. Specifically, PEARL generates fewer draft tokens if they will be rejected (step 1 in the lower part), and more draft tokens if they can be accepted (steps 3 and 4).

Albeit with multiple benefits of this *draft-then-verify* framework, it confronts one significant challenge that may hinder its performance and deployment—the mutual waiting problem. That is, the target model will be idle when the draft model is generating the draft tokens and the draft model will be idle when the target model is verifying the previously drafted tokens. This mutual waiting problem primarily stems from two limitations inherent in speculative decoding: **(i)** the asynchronous execution of the draft and verify phases, which directly results in the mutual waiting problem; and **(ii)** the fixed draft length, which cannot adapt to most decoding steps and thus exacerbate the mutual waiting problem.

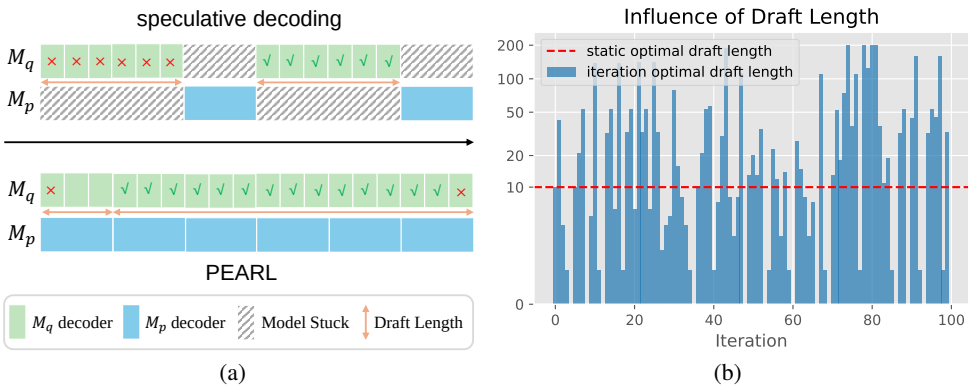
Therefore, in this paper, we seek to answer the question: *Can we draft and verify in parallel and adaptively adjust draft length?* With this consideration, we propose a conceptually simple, flexible, and general framework to boost speculative decoding, namely **P**arallel sp**E**culative decoding with **A**daptive d**R**aft **L**ength (PEARL). Specifically, PEARL consists of two strategies *pre-verify* and *post-verify*: **(i)** *pre-verify* uses the target model to verify the first draft token during drafting phase, which allows the draft model to generate less draft tokens in difficult scenarios; **(ii)** *post-verify* uses the draft model to continue generating draft tokens during verification phase, which provides more draft tokens in simple situations. As shown in the lower part of Figure.1, PEARL effectively alleviates the mutual waiting problem with **parallelism** and **adaptive draft length** via these two strategies. Moreover, we theoretically show that PEARL can **eliminate the burden of tuning the hyperparameter** γ, and the mean accepted tokens of PEARL is **more than** existing *draft-then-verify* works.

Our key contributions can be summarized as follows:

(i) We propose PEARL, a novel inference acceleration framework, which can effectively alleviate the mutual waiting problem with parallelism and adaptive draft length.

(ii) We theoretically derive the optimal window size and the mean accepted tokens of our PEARL, which demonstrates the effectiveness of our PEARL.

(iii) We conduct extensive experiments on various text generation benchmarks, leading to a superior speedup performance up to **4.43**$\times$ and **1.50**$\times$, compared to auto-regressive decoding and speculative decoding, respectively.



(a) (b)

Figure 2: Motivated observations. (a) The time of both drafting phase and verification phase is non-negligible, therefore the asynchronous execution of the draft model and the target model directly incurs the mutual waiting problem. (b) We observe that the optimal draft length changes significantly in different iterations, which exacerbates the mutual waiting problem.

2 Background

Notations. In this paper, we use M_q to denote the draft model and M_p to denote the target model. $M_q(\cdot), M_p(\cdot)$ denotes the logits of the next token of a single forward of M_q, M_p respectively. γ is a hyperparameter to control the window size during speculative decoding. We denote the running speed between M_q and M_p as c, which is defined as the ratio between the time for a single forward of M_p and the time for a single forward of M_q, i.e., $c = T(M_p(\cdot))/T(M_q(\cdot))$.

Speculative decoding. Given an input sequence $\mathbf{x}$ as prefix, a speculative decoding step consists of a drafting phase and a verification phase. During the drafting phase, the draft model M_q is employed to give γ draft tokens $x_1, x_2, ..., x_\gamma$ by running γ times model forward and sample. Here, we denote $M_q(\mathbf{x} + [x_1, ..., x_{i-1}])$ as q_i, then each draft token is given by $x_i \sim q_i, i = 1, ..., \gamma$. During the verification phase, the prefix $\mathbf{x}$ together with γ draft tokens are sent to M_p for verification. The target model M_p inputs $\mathbf{x} + [x_1, ..., x_\gamma]$ and outputs the logits $p_1, p_2, ..., p_{\gamma+1}$. Then SD sequentially verifies x_i via speculative sampling, where the acceptance rate is given by:

$$\alpha_i = \begin{cases} 1 & p_i[x_i] \geq q_i[x_i], \\ \dfrac{p_i[x_i]}{q_i[x_i]} & p_i[x_i] < q_i[x_i], \end{cases} \tag{1}$$

If SD rejects x_i, it will resample a token from $norm(\max(0, p_i - q_i))$, otherwise SD accepts all the draft tokens and samples an additional token from $p_{\gamma+1}$. In this way, each SD step generates tokens with a number of at least 1 and at most $\gamma + 1$, leading to the efficiency acceleration.

Window size and draft length. We emphasize that the window size is a hyperparameter that controls the drafting behavior. Draft length is the number of tokens generated by the draft model in a continuous execution, which is fixed and same as the window size in SD, while draft length is adaptive and may be not equal to window size in PEARL.

3 Methodology

3.1 Motivated Observation

As illustrated in Figure. 2(a), the mutual waiting problem is directly incurred by the asynchronous execution of the draft model and the target model. In our experiments, we observe that the time consumed during the drafting phase and the verification phase is usually non-negligible. Take the instance of Codellama 7B & 34B, at each decoding step, although the running speed of Codellama 7B is almost 3 times faster than Codellama 34B, the total time consumption for generating 6 draft

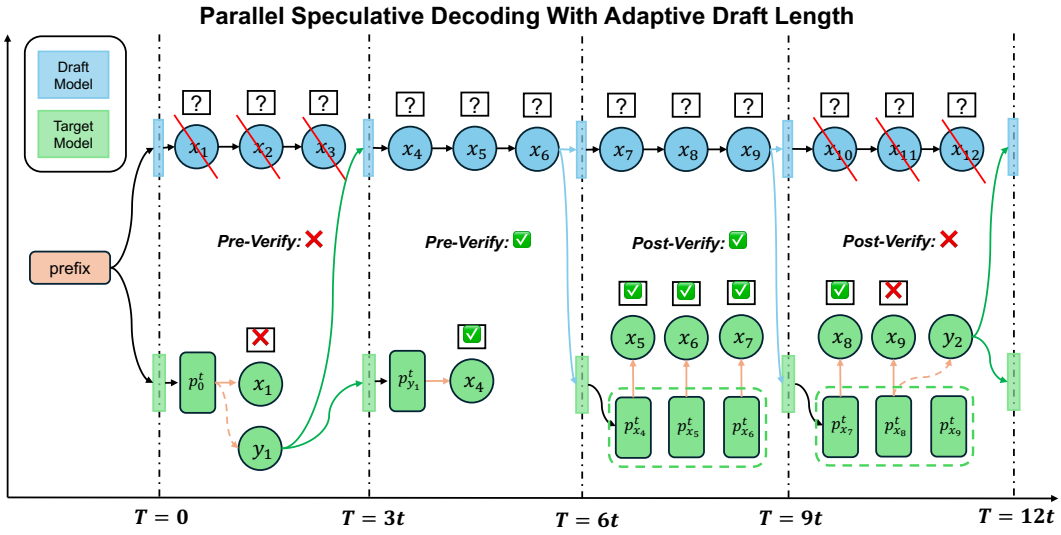


Figure 3: Illustration of our PEARL. At $T = 0$, M_q generates x_1, x_2, x_3 and M_p rejects x_1 with the *pre-verify* strategy. At $T = 3t$, M_p accepts x_4 and switches to the *post-verify* strategy. At $T = 6t$, M_p accepts all draft tokens x_4, x_5, x_6 in the last decoding step, while M_q continues drafting x_7, x_8, x_9. At $T = 9t$, M_p rejects x_9, drops x_{10}, x_{11}, x_{12} and switches to the *pre-verify* strategy. The final output is $[y_1, x_4, x_5, x_6, x_7, x_8, y_2]$.

tokens is even 2 times than the time consumption for one verification step. Therefore, the mutual waiting problem exists **at any timestamp**, and severely affects the acceleration effectiveness of SD.

The asynchronous execution of the draft model and the target model is the direct cause of the mutual waiting problem, which is determined by two requirements of speculative decoding: (1) the drafting phase requires the input prefix to be verified; (2) the verification phase requires the draft model to complete generating draft tokens. This implies the great potential for alleviating the mutual waiting problem through parallelism: if we can remove the two requirements and parallel the drafting phase and the verification phase, a substantial acceleration can be possible.

Another limitation that aggravates the mutual waiting problem is the fixed draft length in SD, which is not appropriate for all the decoding steps. As shown in Figure 2(b), the optimal draft length changes significantly in different iterations. On the one hand, when the optimal draft length is less than the fixed draft length, the draft model will generate meaningless draft tokens that blocks the target model. On another hand, when the optimal draft length is more than the fixed draft length, the draft model could have generated more draft tokens that can be accepted by the target model with a single forward. However, a fixed draft length will interrupt the longer drafting phase and take an additional verification phase, which strengths the mutual waiting problem as well. This motivates our PEARL to further alleviate the mutual waiting problem with adaptive draft length.

Together with the two motivations, we propose two simple and effective strategies, *pre-verify* and *post-verify*. The *pre-verify* removes requirement 2 and allows the target model to verify the first draft token in advance. The *post-verify* removes requirement 1 and allows the draft model to continue generating draft tokens during the verification phase. The two strategies enable parallelism and achieve adaptive draft length to effectively alleviate the mutual waiting problem.

3.2 PRE-VERIFY: VERIFY THE FIRST DRAFT TOKEN IN ADVANCE.

The *pre-verify* strategy aims at removing the requirement that the verification phase requires the draft model to complete generating draft tokens. Therefore, we seek to verify some draft tokens in advance during drafting phase. We delve explicitly into the drafting stage. During the drafting phase, the draft model tries to give γ draft tokens by running γ times model forward. We find that the input of the draft model in γ times forward is $\mathbf{x}$, $\mathbf{x} + [x_1]$, ..., $\mathbf{x} + [x_1, x_2, ..., x_{\gamma-1}]$, respectively.

Only the origin prefix $\mathbf{x}$ can be acquired by the target model for parallel verification. Therefore, we propose to run the target model to output the logits $M_p(\mathbf{x})$ in parallel. In this way, we can verify the first token x_1 before the verification phase. We implement the same lossless verification method following (Leviathan et al., 2023) as illustrated in Section 2.

By applying such a *pre-verify* strategy, we can verify the first draft token before the verification phase. If the first token is rejected, all of the following draft tokens are meaningless and should be dropped. Hence we could skip the verification phase and directly conduct the next drafting phase with the prefix $\mathbf{x} + [y_1]$. If the first token is accepted, all the draft tokens will be sent to the target model in the verification phase. In Figure. 3, at the timestamp of $T = 0$, the draft model generates x_1, x_2, x_3 while the target model outputs p_0^t, rejects the first token x_1 and sample another token y_1. At the timestamp of $T = 3t$, the draft model generates x_4, x_5, x_6 while the target model accepts the first token x_4. Then x_4, x_5, x_6 is sent to the target model in the next verification phase.

3.3 POST-VERIFY: CONTINUE DRAFTING DURING VERIFICATION.

The *post-verify* strategy aims at removing the requirement that the drafting phase requires the input prefix to be verified. However, this assumption brings the limitation that the draft model should be stuck until the target model finishes verification.

Therefore, we discard this assumption and make another assumption: we directly assume that all the draft tokens can be accepted. In this way, We find that when all the γ draft tokens are accepted, sampling a new token from $M_p(\mathbf{x} + [x_1, ..., x_\gamma])$ is not necessary, as the draft model could have generated more draft tokens that can be accepted. Hence we can use the draft model to continue drafting $x_{\gamma+1}, ..., x_{2\gamma}$ during the verification phase.

If all the γ draft tokens are accepted, we can skip the next drafting phase as we already get the draft tokens in the next drafting phase. The last logit $M_p(\mathbf{x} + [x_1, ..., x_\gamma])$ can be used to verify $x_{\gamma+1}$, which is a *"pre-verify"* process as well. In Figure. 3, at the timestamp of $T = 6t$, the target model takes in x_4, x_5, x_6 and outputs $p_{x_4}^t, p_{x_5}^t, p_{x_6}^t$, while the draft model continues to guess next draft tokens x_7, x_8, x_9. Fortunately, all the draft tokens are accepted, and we can directly conduct the next verification phase with prefix $\mathbf{x} + [y_1, x_4, x_5, x_6, x_7, x_8, x_9]$. At the timestamp of $T = 9t$, the target model takes in x_7, x_8, x_9 and outputs $p_{x_7}^t, p_{x_8}^t, p_{x_9}^t$, while the draft model continues to guess the next draft tokens x_{10}, x_{11}, x_{12}. Unfortunately, only x_8 is accepted, and the draft tokens x_{10}, x_{11}, x_{12} will be dropped. Finally, the prefix $\mathbf{x} + [y_1, x_4, x_5, x_6, x_7, x_8, y_2]$ is input to the next drafting phase.

Algorithm 1 Parallel Speculative Decoding with Adaptive Draft Length.

Require: the draft model M_q, the target model M_p, the input prefix $\mathbf{x}$, the max generate tokens L, the window size γ.
Initialization: mode $\leftarrow$ "pre-verify"
while $len(\mathbf{x}) < L$ **do**
  if mode = "pre-verify" **then**
    $\mathbf{x}$, mode $\leftarrow$ Pre-verify($M_q, M_p, \mathbf{x}, \gamma$)
  else
    $\mathbf{x}$, mode $\leftarrow$ Post-verify($M_q, M_p, \mathbf{x}, \gamma$)
  end if
end while

3.4 PEARL: PARALLEL SPECULATIVE DECODING WITH ADAPTIVE DRAFT LENGTH

Take together the two strategies, our PEARL framework consists of a draft model, a target model and two strategies to decode tokens. The two strategies are switched according to the verification results in the previous decoding step. Algorithm. 1 provides a summary of our PEARL. We also provide more details in Algorithm. 2. Note that pre-verify and post-verify strategies are not executed only once in the process of generating a sentence and will be repeatedly switched according to the token acceptance situation during the whole process of generating. Then we show how our PEARL achieves parallelism and adaptive draft length to alleviate the mutual waiting problem.

Parallelism. With the two strategies *pre-verify* and *post-verify*, At any timestamp, the draft model and the target model are running in parallel, which directly breaks the asynchronous execution of the draft model and the target model.

Adaptive draft length. In our PEARL, the drafting process can be seen as segmented drafting process. If the draft model cannot generate any "right" tokens, the *pre-verify* strategy will avoid the

additional drafting process. If the draft model could have generated more "right" tokens, the target model will not interrupt the drafting phase, where the draft model can generate more draft tokens with *post-verify* strategy. Therefore, PEARL can utilize the two simple yet effective strategies to implement adaptive draft length to alleviate the mutual waiting problem.

4 ANALYSIS

In this section, we give some interesting theoretical findings to demonstrate the generalization ability and effectiveness of our PEARL.

4.1 ELIMINATING THE BURDEN OF TUNING γ

During a standard speculative decoding step, the draft model takes γ model forward to decode γ draft tokens and then the target model takes 1 model forward to verify the draft tokens. If γ is set too small, we cannot fully exploit the ability of the draft model, and the realistic speedup will be tiny. If γ is set too large, the extra overhead of running draft model will cover the acceleration of speculative decoding, leading to an undesirable speedup. Hence it is crucial to find an optimal value of γ for considerable acceleration.

Assuming the acceptance rate is α, the optimal value γ' of γ in SD is given as follows:

$$\gamma' = \arg\max_{\gamma} SD(\gamma) = \arg\max_{\gamma} \frac{1-\alpha^{\gamma+1}}{(1-\alpha)(\frac{\gamma}{c}+1)}, \tag{2}$$

where α is expected acceptance rate between the draft model and the target model, and c is the ratio between the time for a single forward of target model and the draft model. However, there lies some issues in finding γ' with this equation. On the one hand, this equation makes an important assumption that the acceptance rate of each token is i.i.d.. Actually, this assumption is only an approximation, leading to difficulties in finding an appropriate γ. On another hand, the value of α may vary in different scenarios, as the draft model and the target model may show different alignments in different scenarios. Therefore, for a specific downstream application, one needs to search an optimal value of γ, which brings severe burden and limits the application of SD.

In our PEARL, γ' can be theoretically found without these issues. We give Theorem. 1 to show that the optimal value of γ is exactly c.

Theorem 1 *Given a draft model M_q and a target model M_p, the optimal value of the window size γ is the ratio of the running speed of the draft model and the target model, i.e.,*

$$\gamma' = \arg\max_{\gamma} \textit{ PEARL}(\gamma) = c. \tag{3}$$

We prove this theorem in Appendix B. Intuitively, increasing or decreasing γ' does not contribute to better efficiency. We conduct experiments in Section 5.4.1 to empirically demonstrate this theorem, which is important for our PEARL to **eliminate the burden of tuning γ.**

4.2 EXPECTATION OF THE NUMBER OF ACCEPTED TOKENS

As our PEARL is based on the standard speculative decoding, we theoretically compare the decoding step of our PEARL and standard SD. Suppose the draft model M_q and the target model M_p are same in the two algorithms. We do not introduce any extra training or fine-tuning, hence the acceptance rate for each token α is same and can be seen as a constant in the following analysis.

We begin our analysis by computing the expectation of the number of accepted tokens within a drafting phase and a verification phase. Note that in our PEARL, if all the draft tokens in a drafting phase are accepted, PEARL will continue drafting more draft tokens. We regard the process that the draft model decodes draft tokens until some tokens are rejected as a drafting phase when computing the expectation.

Theorem 2 *Assuming the acceptance rate of each draft token is α, and α is i.i.d., the expectation of the number of accepted tokens of PEARL is*

$$E(\#accepted\ tokens) = \frac{1}{1-\alpha} + 1. \tag{4}$$

We prove this theorem in Appendix B. We give a more intuitive explanation of this theorem: given a prefix, if the draft model M_q can decode δ draft tokens at most that can be accepted by the target model, then M_q will continue to decode δ tokens without being interrupted by the target model verification. It can be explained as an adaptive "γ" for each prefix to achieve better speedup.

Note that the expectation of accepted tokens of other *draft-then-verify* works is $\frac{1-\alpha^{\gamma+1}}{1-\alpha}$. With Theorem. 2, it is easily to show that **the expectation of accepted tokens of PEARL is more than standard SD**, i.e., $\frac{1}{1-\alpha} + 1 \geq \frac{1-\alpha^{\gamma+1}}{1-\alpha}$. From this perspective, we can demonstrate the effectiveness of our PEARL. We will conduct experiments in Section 5.4.2 to show the empirical results of mean accepted tokens in both standard SD and our PEARL.

5 Experiments

5.1 Experimental Setup

Tasks and Datasets. We conduct experiments on various text generation tasks to evaluate the effectiveness of our PEARL, including HumanEval (code generation) (Chen et al., 2021), GSM8K & MGSM (multilingual arithmetic reasoning) (Cobbe et al., 2021; Shi et al.), and MT-bench (multi-round dialogue) (Zheng et al., 2024). More details can be found in Appendix C.1.

Evaluation Details. We evaluate the effectiveness of our PEARL with some state-of-the-art LLM families, including CodeLlama (Roziere et al., 2023), Deepseek-Coder (Guo et al., 2024), Llama 2 (Touvron et al., 2023) and Llama 3.1 (Dubey et al., 2024). In our experiments, the models with size less than 7B are used as the draft models and the models with size greater than 33B are used as the target models. We report the walltime speedup ratio as the metric. Additional evaluation details are provided in Appendix C.2 and C.3.

Baseline Methods. We implement *five training-free* inference acceleration methods as our baselines. **(i) Speculate decoding:** standalone SD methods (Leviathan et al., 2023; Chen et al., 2023) resort to a draft model to draft future tokens and then verify them in parallel. **(ii) Ouroboros:** ouroboros (Zhao et al., 2024) proposes phrase candidate pool from the verification process to generate more precise and longer drafts. **(iii) Lookahead Decoding:** look ahead decoding (Fu et al., 2024) caches the generation trajectory (n-grams) as drafts to reduce the number of total decoding steps. **(iv) Distillspec:** DistillSpec (Zhou et al., 2023) distillates and derives a better aligned and compact draft model. **(v) Assisted generation:** assisted generation employs a heuristic approach to determine the number of draft tokens in the next iteration, based on the verification results of tokens generated by the draft model in the previous round.

Table 1: Experiment results on the code generation task. Part of the results of Lookahead Decoding and Ouroboros are taken from (Zhao et al., 2024). We **bold** the best results for each model combination. * Some results of ouroboros and lookahead decoding are reproduced in their official implementation. Other results are reproduced in our implementation. The symbol '-' denote that the methods do not support current model configuration.

Method	CodeLlama 7&34B	CodeLlama 7&70B	Llama2 7&70B	Llama3.1 8&70B	DeepSeek 1.3&33B	DeepSeek 6.7&33B
Auto Regressive	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
Speculative Decoding	1.76×	3.03×	2.35×	2.60×	2.32×	1.94×
Ouroboros	2.14×	*3.28×	*2.10×	-	*3.25×	*2.66×
Lookahead Decoding	1.72×	*1.57×	*1.80×	-	*1.82×	*1.82×
Assisted Generation	1.37×	2.49×	2.27×	2.72×	1.88×	1.52×
Ours	**2.48**×	**4.43**×	**3.29**×	**3.87**×	**3.48**×	**2.79**×

5.2 Main results.

Table 1 shows that PEARL significantly outperforms vanilla speculative decoding, ouroboros, and lookahead decoding in all backbone model configurations on the HumanEval dataset. Specifi-

Table 2: Multi-language experiment results using Llama 3.1 8B&70B on GSM8K and MGSM (Cobbe et al., 2021; Shi et al.). We **bold** the best results for each language.

Method	English (GSM8K)	Bengali	German	Spanish	French	Japanese	Russian	Swahili	Tegulu	Thai	Chinese	Avg.
AR	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
SD	2.48×	2.69×	2.77×	2.64×	2.71×	2.71×	2.72×	2.81×	2.65×	2.71×	2.78×	2.70×
Ours	**3.82**×	**3.94**×	**4.00**×	**3.81**×	**3.76**×	**3.94**×	**3.85**×	**4.18**×	**4.10**×	**3.93**×	**4.06**×	**3.95**×

Table 3: Experiment results using Llama2 7B&70B and Llama 3.1 8B&70B on MT-bench (Zheng et al., 2024). We **bold** the best results for each model configuration.

Model Configuration	Method	Writing	Roleplay	Reasoning	Math	Coding	Extraction	Stem	Humanities	Avg.
Llama 2 7B&70B	AR	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
	SD	1.70×	1.73×	1.96×	2.00×	1.93×	2.14×	1.87×	1.81×	1.89×
	Ours	**2.40**×	**2.45**×	**2.85**×	**2.79**×	**2.67**×	**2.92**×	**2.58**×	**2.50**×	**2.64**×
Llama 3.1 8B&70B	AR	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
	SD	2.29×	2.24×	2.66×	2.81×	2.35×	2.64×	2.22×	2.12×	2.42×
	Ours	**3.49**×	**3.35**×	**3.92**×	**4.06**×	**3.55**×	**3.95**×	**3.34**×	**3.05**×	**3.59**×

cally, PEARL can achieve up to 4.43 × speed up compared with vanilla auto-regressive methods. These experimental results demonstrate that PEARL effectively addresses the mutual waiting issue, thereby achieving significant inference acceleration results compared to methods based on the traditional draft-then-verify framework. PEARL can also achieve significant inference acceleration in conventional natural language processing tasks, whereas vanilla speculative decoding yield only limited improvements regarding acceleration. As shown in Tables 2 and 3, PEARL leads to superior performances on the GSM8K and MT-bench datasets as well.

5.3 Ablation Studies

To provide more insights of the two proposed strategies, we conduct the ablation study. We denote PEARL without *pre-verify* as PEARL *w/o pre-verify* and PEARL without *post-verify* as PEARL *w/o post-verify* and present the main results of ablation studies.

As shown in Table 4, the absence of any strategy of PEARL results in a performance degradation of the entire framework. The absence of the *post-verify* strategy exhibits a more pronounced impact on the performance of PEARL than the *pre-verify* strategy. Intuitively, the *pre-verify* strategy makes more contributions when the acceptance rate is relatively low, while the *post-verify* strategy makes more contributions when the two models are aligned. Therefore, the two strategies are complementary and accelerate inference together. In our experiments, all the models combinations show great alignment, which leads to the superiority of the *post-verify* strategy.

5.4 Case Studies

5.4.1 Optimal Results of the Window Size γ

As mentioned in Section 4.1, PEARL eliminates the need for tuning the hyperparameter of window size and can theoretically predetermine the optimal value of γ. In this section, we conduct experiments with various gamma values in our PEARL to demonstrate the impact of the optimal γ on accelerating LLM inference. As shown in Tables 5 and 6, we can observe that with the predetermined optimal γ, PEARL achieves the maximum inference acceleration. In certain cases involving suboptimal γ, the inference acceleration improvement in PEARL is less significant, underscoring the importance of predetermining the gamma value.

5.4.2 Mean Accepted Tokens

In Section 4.2, we theoretically demonstrate that the expected number of accepted tokens in PEARL exceeds that of the vanilla SD method. To further illustrate the real mean accepted tokens in PEARL under real-world complex conditions, we conduct experiments on the HumanEval, GSM8K, and MT-Bench datasets. As shown in Table 7, we still empirically observe that that PEARL obtains more accepted tokens compared to vanilla SD methods, which further demonstrates the effectiveness of the PEARL framework. Specifically, PEARL achieves the max number of mean accepted tokens to

Table 4: Ablation results of PEARL on HumanEval and GSM8K datasets.

	HumanEval			GSM8K
Methods	**CodeLlama 7B&34B**	**CodeLlama 7B&70B**	**DeepSeek 1.3B&33B**	**Llama 2 7B&70B**
PEARL *w/o pre-verify*	2.21×	3.53×	3.19×	2.51×
PEARL *w/o post-verify*	1.64×	2.57×	2.37×	2.15×
PEARL	**2.35**×	**3.79**×	**3.48**×	**2.87**×

Table 5: Optimal γ of different model combinations on HumanEval. (unit: tok/sec)

γ	CodeLlama 7&34 (c=3)	CodeLlama 7&70 (c=5)	DeepSeek 6.7&33 (c=3)
2	33.25	16.28	30.82
3	**46.06**	23.14	**48.46**
4	44.12	29.65	47.22
5	44.93	**40.72**	46.91
6	41.83	35.39	44.36

Table 6: Optimal γ for different tasks of Llama 2 7B&70B. (c=5)

γ	HumanEval	GSM8K	MT-Bench
3	20.39	18.23	17.67
4	24.58	21.81	20.69
5	**30.34**	**26.47**	**24.25**
6	28.02	24.59	22.71
7	28.09	24.23	22.54

Table 7: Comparison of mean average accepted tokens of vanilla SD methods and PEARL.

Methods	**CodeLlama 7B&34B**	**CodeLlama 7B&70B**	**DeepSeek 1.3B&33B**	**DeepSeek 6.7B&33B**
SD	5.27	8.32	7.23	5.69
Ours	**27.95**	**26.53**	**29.65**	**39.90**

39.9, which significantly outperforms vanilla SD methods by a large margin. Note that the mean average accepted tokens per second (mean AAT) and the speed ratio c between the draft model and the target model both influence the final speed-up results. For example, in the case of Deepseek 6.7B and 33B, the draft model runs approximately three times faster than the target model. Even if the mean AAT approaches infinity, where all tokens are generated by the 6.7B model, the theoretical maximum speed-up would be capped at 3x. Consequently, with a mean AAT of 40, PEARL achieves a 2.75x speed-up, which is close to this theoretical optimum. These results demonstrate that our PEARL can fully exploit the inference ability of the draft model for further acceleration.

5.4.3 Limited GPU Resources Scenarios

We discuss our PEARL in the limited GPU resource scenarios, which we refer to "co-locate" setting or resource competitions (RC). The key problem lies in the nature of GPU hardware design—two running processes on the same GPU will compete for GPU resources, which may lead to slowdowns.

Generally, in real-world LLM applications, the large-scale target model is usually placed with more than 1 GPU to handle more requests and long context inference, while the small-scale draft model only needs 1 GPU for inference. In this case, we can apply pipeline parallelism (PP) to serve the target model with multiple GPUs. Inspired by this observation, we propose an improved version of PEARL to effectively utilize GPU computation resources with PP without resource competitions. The key idea is to transfer the computation of the draft model to another GPU when the target model is running on a specific GPU. Specifically, we transfer the first $\lceil \frac{\gamma}{2} \rceil$ draft token generation to the last device, while the last $\lfloor \frac{\gamma}{2} \rfloor$ draft tokens are generated with the first device. As the computation of the target model is conducted sequentially with multiple GPUs, this could effectively utilize the GPU resources to avoid RC. We conduct some experiments in Table 8 and find that this strategy allows PEARL to retain $89\% \sim 99\%$ of its original performance, demonstrating the effectiveness of our PEARL in such conditions. We provide detailed implementation of this strategy in Appendix E.

6 Related Work

Transformer inference acceleration. There exists extensive works for transformer inference acceleration. This includes efforts of model compression (Zhu et al., 2023), efficient architecture design (Chitty-Venkata & Somani, 2022), and hardware optimization and implementation (Dao et al.,

Table 8: Comparisons of Llama models for PEARL on MT-bench with and without RC scenario.

Models	Writing	Roleplay	Reasoning	Math	Coding	Extraction	Stem	Humanities	Avg.
Llama 2 7b&70b	22.10	22.47	26.16	25.74	24.63	26.11	23.84	23.09	24.28
Llama 2 7b&70b (RC)	19.88	20.24	25.06	24.47	23.47	25.79	21.55	22.03	22.83
performance retain	89.95%	90.08%	95.80%	95.07%	95.29%	98.77%	90.39%	95.41%	94.03%
Llama 3.1 8b&70b	31.23	30.08	35.09	36.59	31.95	34.60	30.06	27.51	32.14
Llama 3.1 8b&70b (RC)	29.65	27.54	35.01	36.31	29.85	33.99	26.77	26.10	30.78
performance retain	94.94%	91.56%	99.77%	99.23%	93.43%	98.24%	89.06%	94.87%	95.77%

2022). Model compression methods such as quantization (Choi et al., 2018), knowledge distillation (Hinton et al., 2015), and structure pruning (Han et al., 2015) aim at reducing the number of computational operations. Efficient architecture design is proposed to develop lightweight transformer architectures. Hardware optimization and implementation is proposed for efficient execution to fully exploit the hardware devices. These methods have achieved great success, while they are orthogonal to speculative decoding algorithms, which can be integrated for further speedup.

Draft-then-verify framework. While SD exhibits great acceleration effectiveness and lossless generalization quality, it remains a challenge to find a compact draft model with high distribution alignment. Some works focus on removing the necessity of the draft model. Self-speculative decoding (Zhang et al., 2023) proposes to skip some intermediate layers of the target model for drafting. Medusa (Cai et al., 2024) adds extra decoding heads at the top of the target model to generate drafts. Lookahead decoding(Fu et al., 2024) caches the generation trajectory (n-grams) as the drafts. Eagle (Li et al., 2024) employs an additional transformer decoder layer to generate drafts at the feature level. Glide (Du et al., 2024) reuses the kv cache from the target model to decode more accurate draft tokens. DistillSpec (Zhou et al., 2023) utilizes distillation method to identify a compact draft model. Ouroboros (Zhao et al., 2024) combines the standard SD and lookahead decoding to generate more precise and longer drafts. Besides these works, SpecInfer (Miao et al., 2023) proposes tree attention, which is widely used to verify more drafts and increase the acceptance rate. However, all of them do not address the parallelism issue. From this perspective, our PEARL is orthogonal to these methods and can be integrated with these methods, which is left as a future work.

7 Conclusion and Future Work

Limitations and broader impact. As our PEARL is a parallel acceleration framework, it remains a challenge to schedule the GPU resources to avoid resource competitions, which may potentially increase power consumption. We affirm our commitment to contributing positively to society, avoiding harm, and upholding honesty and trustworthiness. We appropriately cite the previous methods and datasets we use, and ensure that all data involved is fully public, with no private data being utilized. Furthermore, we are committed to correctly maintaining the inference acceleration techniques we have developed, without incurring any form of discrimination.

Conclusion. In this paper, we propose a novel inference acceleration framework, called PEARL, which significantly improves LLM inference efficiency. PEARL consists of two simple and effective strategies, i.e., *pre-verify* and *post-verify*, which effectively alleviates the mutual waiting problem with parallelism and adaptive draft length. Moreover, We theoretically derive the optimal window size and the mean accepted tokens of our PEARL, which demonstrates the effectiveness of our PEARL. Extensive experiments demonstrate that our proposed PEARL outperforms existing state-of-the-art methods on various text generation benchmarks.

Future work. For future research, we aim to integrate PEARL with existing accelerated inference methods to explore more efficient and resource-friendly acceleration approaches for LLM inference. Hopefully, PEARL will facilitate the future development of LLM inference acceleration.

References

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

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

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

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

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

Krishna Teja Chitty-Venkata and Arun K Somani. Neural architecture search survey: A hardware perspective. *ACM Computing Surveys*, 55(4):1–36, 2022.

Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

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

Cunxiao Du, Jing Jiang, Xu Yuanchen, Jiawei Wu, Sicheng Yu, Yongqi Li, Shenggui Li, Kai Xu, Liqiang Nie, Zhaopeng Tu, et al. Glide with a cape: A low-hassle method to accelerate speculative decoding. *arXiv preprint arXiv:2402.02082*, 2024.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

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

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

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Diksha Khurana, Aditya Koli, Kiran Khatter, and Sukhdev Singh. Natural language processing: State of the art, current trends and challenges. *Multimedia tools and applications*, 82(3):3713–3744, 2023.

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

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

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

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, et al. Language models are multilingual chain-of-thought reasoners, 2022. *URL https://arxiv. org/abs/2210.03057*.

Anthropic Team. The claude 3 model family: Opus, sonnet, haiku. 2024. URL `https://api.semanticscholar.org/CorpusID:268232499`.

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

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

Weilin Zhao, Yuxiang Huang, Xu Han, Chaojun Xiao, Zhiyuan Liu, and Maosong Sun. Ouroboros: Speculative decoding with large model enhanced drafting. *arXiv preprint arXiv:2402.13720*, 2024.

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

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. Distillspec: Improving speculative decoding via knowledge distillation. *arXiv preprint arXiv:2310.08461*, 2023.

Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. A survey on model compression for large language models. *arXiv preprint arXiv:2308.07633*, 2023.

A ALGORITHM OF PEARL

Here, we give the whole algorithm of our PEARL in detail in Algorithm. 2.

Algorithm 2 Parallel Speculative Decoding with Adaptive Draft Length.

Input: the draft model M_q, the target model M_p, the input prefix $\mathbf{x}$, the max generate tokens L, the window size γ.

▷ The *pre-verify* strategy is used first.

Initialization: mode ← "pre-verify"
while $len(\mathbf{x}) < L$ **do**
  if mode = "pre-verify" **then**
    ▷ Pre-verify strategy
    for $i = 1$ to γ **do**
      $q_i \leftarrow M_q(\mathbf{x} + [x_1, ..., x_{i-1}])$
      $x_i \sim q_i$
    end for
    ▷ running the target model in parallel to verify the first draft token in advance.
    $p \leftarrow M_p(\mathbf{x})$
    if $r \sim U(0,1) \leq \frac{p[x_1]}{q_1[x_1]}$ **then**
      ✓ accept the first token
      $\mathbf{x} \leftarrow \mathbf{x} + [x_1, ..., x_\gamma]$
      mode ← "post-verify"
    else
      × reject the first token
      $y \sim norm(max(0, p - q_1))$
      $\mathbf{x} \leftarrow \mathbf{x} + [y]$
      mode ← "pre-verify"
    end if
  else
    ▷ Post-verify strategy
    $\mathbf{x}, [x_1, x_2, ..., x_\gamma] \leftarrow \mathbf{x}$ ▷ split the prefix to get the last γ draft tokens
    for $i = \gamma + 1$ to 2γ **do**
      ▷ running the draft model in parallel to continue drafting.
      $q_i \leftarrow M_q(\mathbf{x} + [x_1, ..., x_{i-1}])$
      $x_i \sim q_i$
    end for
    $p_1, p_2, ..., p_\gamma \leftarrow M_p(\mathbf{x} + [x_1]), M_p(\mathbf{x} + [x_1, x_2]), ..., M_p(\mathbf{x} + [x_1, ..., x_\gamma])$
    retrival $q_1, q_2, ..., q_\gamma$ from the cache
    $r_1 \sim U(0,1), ..., r_\gamma \sim U(0,1)$
    $n \leftarrow \min(\{i - 1 | 1 \leq i \leq \gamma, r_i > \frac{p_i[x_i]}{q_i[x_i]}\} \cup \{\gamma\})$
    if $n = \gamma$ **then**
      ✓ accept all draft tokens
      $\mathbf{x} \leftarrow \mathbf{x} + [x_1, ..., x_{2\gamma}]$
      mode ← "post-verify"
    else
      × reject someone
      $y \sim norm(max(0, p_{n+1} - q_{n+1}))$
      $\mathbf{x} \leftarrow \mathbf{x} + [x_1, ..., x_n, y]$
      mode ← "pre-verify"
    end if
  end if
end while

B PROOF OF THEOREM 1 AND THEOREM 2

As illustrated in Section 4, we have two theorems to demonstrate the effectiveness of our PEARL. Here we give the proof of these two theorems.

Theorem 1 *Given a draft model M_q and a target model M_p, the optimal value of the window size γ is the ratio of the running speed of the draft model and the target model, i.e.,*

$$\gamma' = \arg\max_{\gamma} \textit{PEARL}(\gamma) = c. \tag{5}$$

Proof. We discuss the situation that $\gamma < c$ and $\gamma > c$.

If γ is smaller than c, the draft model will wait for the target model until it finishes the verification. The draft model could have generated more draft tokens to be verified without extra time consumed, That is to say, the situation $\gamma < c$ decreases the number of verified tokens, where PEARL needs to take more steps for inference.

If γ is larger than c, the target model will wait for the draft model until it finishes generating draft tokens. Theoretically, the drafting phase takes $\max(\gamma t, ct)$ and the verification phase takes ct, while the verification phase can verify γ tokens. Therefore, the verification speed is $\frac{\max(\gamma t, ct)+ct}{\gamma}$. As $\gamma > c$, the verification speed is $\frac{(\gamma+c)}{\gamma}t$. Obviously, the optimal value of γ is c.

□

Theorem 2 *Assuming the acceptance rate of each draft token is α, and α is i.i.d., the expectation of the number of accepted tokens of PEARL is*

$$E(\#accepted\ tokens) = \frac{1}{1-\alpha} + 1. \tag{6}$$

Proof. While vanilla speculative decoding can decode $\gamma + 1$ tokens at most, our PEARL can decode infinity tokens due to the adaptive draft length, therefore the expectation of accepted tokens is given by:

$$\begin{aligned} E &= \sum_{k=0}^{\infty} kP(acc = k) \\ &= (1-\alpha)\sum_{k=0}^{\infty} k\alpha^k \end{aligned} \tag{7}$$

Let $S = \sum_{k=0}^{\infty} k\alpha^k$, $\alpha S = \sum_{k=1}^{\infty}(k-1)\alpha^k$, we have:

$$\begin{aligned} E = (1-\alpha)S &= S - \alpha S \\ &= \sum_{k=0}^{\infty} k\alpha^k - \sum_{k=1}^{\infty}(k-1)\alpha^k \\ &= \frac{1}{1-\alpha} \end{aligned} \tag{8}$$

Counting the additional token generated by the target model, the expectation is:

$$E = \frac{1}{1-\alpha} + 1 \tag{9}$$

□

C EVALUATION DETAILS

C.1 DATASET CONFIGURATIONS

In our experiments, we evaluate the effectiveness of our PEARL on 4 categories of text generation tasks, including code generation, arithmetic reasoning, multilingual inference and multi-round dialogue. For the code generation task, we employ HumanEval (Chen et al., 2021), a famous code generation benchmark which is composed of 164 entries. For arithmetic reasoning and multilingual inference, we employ GSM8K and MGSM (Cobbe et al., 2021; Shi et al.) as the evaluation benchmark. As the GSM8K is the English version of MGSM, we report their results in the same table. For GSM8K, we sample the first 100 entries for evaluation. For other 10 categories in MGSM, we select 10 entries for each language. For multi-round dialogue, we employ MT-bench(Zheng et al., 2024) as the benchmark. The maximum generation lengths of these tasks are respectively set to 1024, 256, 256 and 256.

C.2 MODEL CONFIGURATIONS

We select some representative models for evaluation, including Llama 2 Touvron et al. (2023), Codellama Roziere et al. (2023) and Deepseek-Coder Guo et al. (2024). We summarize the model configuration in Table 9. In our experiments, all models are loaded in the precision of bfloat-16. Our PEARL does not introduce any additional training, and directly uses these models to evaluate our algorithm. The running speed is measured on the code generation tasks.

Table 9: Detailed model configurations.

Models	**Layers**	**dim**	**FFN dim**	**speed (tok/s)**
Codellama-7B	32	4096	11008	49.34
Codellama-34B	48	8192	22016	18.58
Codellama-70B	80	8192	28672	9.20
Deepseek-1.3B	24	2048	5504	63.20
Deepseek-6.7B	32	4096	11008	50.05
Deepseek-33B	62	7168	19200	17.37
Llama-2-7B	32	4096	11008	49.94
Llama-2-70B	80	8192	28672	9.22
Llama-3.1-8B	32	4096	14336	44.37
Llama-3.1-70B	80	8192	28672	9.00

C.3 EVALUATION DETAILS

All of our experiments including latency measurement, ablation studies, and case studies are conducted on NVIDIA A100-SXM4-80G GPUs. For models with size of 1.3B and 7B, we put them on a single A100, while 34B models are deployed on 2 A100, 70B models are deployed on 3 A100. For inference, we use batch size 1, which is commonly used in other speculative decoding works. For the compared baselines, including Lookahead decoding and Ouroboros, we reproduce the results of them on the code generation tasks with the default parameters as described in their paper or code. When evaluating these methods, the model configuration and GPU usage is the same as our PEARL.

Table 10: The number of model runs of the draft model and the target model with different model configurations on HumanEval

	Draft Model (SD)	Target Model (SD)	Draft Model (PEARL)	Target Model (PEARL)
Deepseek 1.3B&33B	140500	35125	181864 (1.29×)	45466 (1.29×)
Deepseek 6.7B&33B	128973	42991	174855 (1.35×)	58285 (1.36×)
Codellama 7B&34B	132054	44018	181020 (1.37×)	60340 (1.37×)
Codellama 7B&70B	151960	30392	198370 (1.30×)	39674 (1.30×)
Llama2 7B&70B	175460	35092	248720 (1.41×)	49744 (1.42×)

As our PEARL is a parallel inference acceleration framework, we implement the parallel algorithm in accelerate, which can be further optimized with other parallel techniques. We leave this as a potential future work to acquire more acceleration.

D FORWARD TIMES COMPARISON OF PEARL AND SD METHODS

Considering that PEARL is a parallel framework, both the draft model and the target model are running simultaneously at all times. Therefore, we measure the number of model runs for PEARL compared to the traditional SD method to provide a more comprehensive perspective in Table 10. The results show that our PEARL exhibits relatively more forward times of both the draft model and the target model compared to traditional SD. As our PEARL is a parallel inference framework, which executes the draft model and the target model in parallel at any timestamp, it naturally increases the forward times of the target model and leads to more power consumption. However, the additional inference time occurs at another process, which will not affect the multi-user throughput.

E EXPERIMENT RESULTS UNDER LIMITED GPU RESOURCES

Although our PEARL parallels the draft model and the target model at the algorithmic level, it still remains a challenge for deployment at the hardware level in the GPU-constrained scenarios, which we refer to "co-locate" setting or resource competitions (RC). The key problem lies in the nature of GPU hardware design —- two running processes on the same GPU will compete for GPU resources, which leads to significant slowdowns.

However, in the real-world LLM applications, the large-scale target model is usually placed with more than 1 GPU to handle more requests and long context inference, while the small-scale draft model only needs 1 GPU for inference. In this case, pipeline parallelism (PP) is the most common solution to serve the target model with multiple GPUs, which distributes the parameters to different GPUs and conducts computations sequentially with these GPUs.

Inspired by this observation, we propose an improved version of PEARL to effectively utilize GPU computation resources with PP without resource competitions. The key idea is to transfer the computation of the draft model to another GPU when the target model is running on a specific GPU. Specifically, we transfer the first $\lceil \frac{\gamma}{2} \rceil$ draft token generation to the last device, while the last $\lfloor \frac{\gamma}{2} \rfloor$ draft tokens are generated with the first device. As the computation of the target model is conducted sequentially with multiple GPUs, this method could effectively utilize the GPU resources to avoid resource competition.

Take an instance of $c = 5$, the target model is placed with $g = 4$ GPUs, we denote the time for a target model forward as t, and the time that the target model runs at GPU 0 is $\frac{t}{4}$. To analyze the GPU utilization in details, we split t into $gc = 20$ steps, where each step $\eta = \frac{t}{20}$. During one target model forward, the occupied GPU number in 20 steps is given by:

$$M_p: \quad 0,0,0,0,0;\ 1,1,1,1,1;\ 2,2,2,2,2;\ 3,3,3,3,3; \tag{10}$$

Then we can further analyze the occupied GPU number of the draft model with proposed methods. First, as the draft model can generate c tokens in 20 steps, it only needs 4 steps to generate 1 draft token. Taking $\lceil \frac{\gamma}{2} \rceil = 3, \lfloor \frac{\gamma}{2} \rfloor = 2$, the first $3 \times 4 = 12$ steps of the draft model will occupy the GPU 3, while the last $2 \times 4 = 8$ steps of the draft model will occupy the GPU 0. Therefore, the occupied GPU number of the draft model in 20 steps is given by:

$$M_q: \quad 3,3,3,3;\ 3,3,3,3;\ 3,3,3,3;\ 0,0,0,0;\ 0,0,0,0; \tag{11}$$

In this way, the draft model and the target model will occupy different devices at each step, which effectively avoid the resource competition. However, in the real-world settings, moving the draft model from the last device to the first device is non-trivial and costly. As a compromise, We propose to load the draft model both at the first device and the last device. During inference process, we only move the intermediate KV Cache from the last device to the first device. Many KV Cache compression methods can help further reduce the cost.

To further evaluate the effectiveness of this method, we conduct some experiments in Table 8. We found that this strategy allows PEARL to retain $89\% \sim 99\%$ of its original performance, demonstrating the effectiveness of our PEARL in such conditions.