Amphista: Accelerate LLM Inference with Bi-directional Multiple Drafting Heads in a Non-autoregressive Style

Anonymous ACL submission

Abstract

Large Language Models (LLMs) inherently use autoregressive decoding, which lacks parallelism in inference and results in significantly slow inference speeds, especially when hardware parallel accelerators and memory bandwidth are not fully utilized. In this work, we propose **Amphista**, a speculative decoding algorithm that adheres to a non-autoregressive decoding paradigm. Owing to the increased parallelism, our method demonstrates higher efficiency in inference compared to autoregressive methods. Specifically, **Amphista** models an *Auto-embedding Block* capable of parallel inference, incorporating bi-directional attention to enable interaction between different drafting heads. Additionally, **Amphista** implements *Staged Adaptation Layers* to facilitate the transition of semantic information from the base model's autoregressive inference to the drafting heads' non-autoregressive speculation, thereby achieving paradigm transformation and feature fusion. We conduct a series of experiments on a suite of Vicuna models using MT-Bench and Spec-Bench. For the Vicuna 33B model, **Amphista** achieves up to **2.75**× and **1.40**× wall-clock acceleration compared to vanilla autoregressive decoding and Medusa, respectively, while preserving lossless generation quality.

1 Introduction

Generative large language models (LLMs) have achieved significant breakthroughs in language processing by scaling the transformer decoder block, revealing a potential path towards AGI (Artificial General Intelligence) (OpenAI, 2022). However, during the decoding process of LLMs, the temporal dependency inherent in autoregressive next-token prediction, coupled with the massive parameter count of foundational models, leads to markedly low inference efficiency, characterized by high latency per token and low throughput per second.

In this context, acceleration during inference has become a burgeoning research area. Speculative decoding (Stern et al., 2018; Chen et al., 2023) uses a draft model for preliminary multi-step speculative inference and a target model to verify the speculative predictions, emerging as a very promising algorithmic strategy. Notably, by employing a rejection sampling strategy (Leviathan et al., 2023), the generation quality and accuracy of the speculate-and-verify framework are consistent with those of the base model, making speculative decoding a lossless acceleration framework. Medusa decoding (Cai et al., 2024) innovatively uses the base model's last hidden states to implement a multi-heads inference framework. It has been widely adopted due to its significant acceleration effect and simple structure.

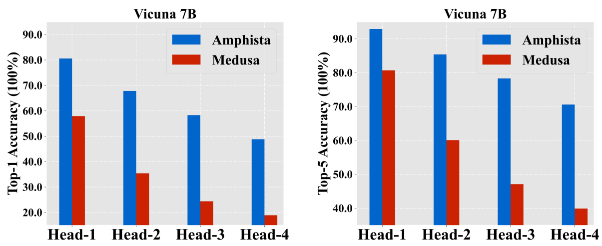

Figure 1: Top-1/5 accuracy for different heads of Medusa and our Amphista. We perform testing with randomly sampled 5% sharegpt conversation data. Amphista far outperforms Medusa in terms of head accuracy, especially for the latter two heads.

However, based on our experiments, as shown in Figure 1, we find that except for the first head, Medusa heads' prediction accuracy is relatively low, which affects the acceleration performance on downstream tasks. To address inaccuracies and ensure the parallel inference capability of the drafting heads, we first propose the Auto-embedding Block, which incorporates a bi-directional self-attention module (Vaswani et al., 2017) following MLPs' activation (see Figure 2). This structure allows preceding drafting heads to attend to subsequent heads and, more importantly, enables backward drafting heads to benefit from the information provided by preceding heads. It equips drafting heads to better represent contextual information, thereby improving the acceptance rate of their predictions. Moreover, this is a non-autoregressive modeling structure that achieves lower drafting latency compared to an autoregressive approach.

Additionally, we observe a gap between the autoregressive base model and the non-autoregressive draft model in terms of token prediction paradigms. To bridge this paradigm gap and further enhance feature representations across different drafting heads, we introduce the staged adaptation layers. These layers serve as an adaptive module between the base model and the drafting model, facilitating the transformation and integration of features. Through their adaptation, the semantically enriched feature is then input into the auto-embedding block after MLP activations. This process significantly aids the bi-directional attention mechanism in fusing features across different heads, ultimately improving the acceptance rate and translating into a noticeable wall-clock time speedup.

Finally, we aim to better align the entire drafting model with the base model. To enhance adaptation with minimal computational overhead, a sampled token from the base model's last prediction step is introduced to the staged adaptation layers. This key integration unites Amphista and the base model more effectively, thus enabling seamless inference acceleration with a significant improvement.

To summarize, our contributions are as follows:

- We propose Amphista, a cost-efficient non-autoregressive inference acceleration framework based on Medusa, enabling bi-directional interaction (Auto-embedding) among different heads during the drafting phase.

- To bridge the token prediction paradigm gap from autoregressive to non-autoregressive modeling and to further enhance the auto-embedding block's representation, we introduce staged adaptation layers to adapt information from the base model's hidden states to different drafting positions in two stages. Additionally, we introduce a sampled token to better align the draft and target models without incurring much overhead.

- We evaluate a suite of foundation models of various sizes. The experimental results show that Amphista significantly outperforms Medusa in both acceptance rate and speed-up across different generation tasks. Notably, our method achieves better gains on larger foundational models, demonstrating a substantial scaling property.

2 Preliminaries

In this section, we introduce some preliminary background related to our work as follows:

Speculative Decoding. Speculative execution is widely utilized in the field of computer architecture and has been successfully applied to LLM decoding algorithm recently (Leviathan et al., 2023; Chen et al., 2023; Stern et al., 2018). The core idea is to leverage a small, lower-quality model (draft model) together with a large, higher-quality model (target model) to accelerate token generation. Concretely, in each decoding step, the algorithm first uses the draft model to autoregressively generate a sequence of future tokens. These drafted tokens are then verified by the target model in a single forward pass. During the verification process, a certain strategy is applied to determine which tokens are accepted by the target model and which are rejected and discarded. Previous work (Leviathan et al., 2023) has theoretically and empirically demonstrated that the token output distribution of speculative decoding is consistent with the autoregressive generation of original target model, but with fewer decoding steps, thus enhancing generation efficiency.

Medusa Decoding. Medusa Decoding (Cai et al., 2024) represents an efficient speculative decoding algorithm that adheres to the draft-and-verify principle. Specifically, Medusa decoding employs several independent MLP layers as drafting heads, which are integrated with the base model to form a unified architecture. During each decoding iteration, the base model's lm_head is used to sample the token at the next-0 position. Concurrently, the i-th MLP head predicts the token at the next-i position. After the generation of these drafting tokens, the base model's forward pass is employed to verify and determine whether to accept or reject these tokens. By utilizing simple MLP layers as drafting heads, Medusa effectively balances computational overhead and prediction accuracy, thereby achieving significant acceleration. Hydra (Ankner et al., 2024), which is a subsequent state-of-the-art optimization based on Medusa, transforms the independent MLP heads into sequentially dependent MLP heads, further enhancing the predictive accuracy.

Tree Attention. Tree attention (Miao et al., 2024; Cai et al., 2024) is proposed to calculate attention scores for multiple draft candidates in parallel.

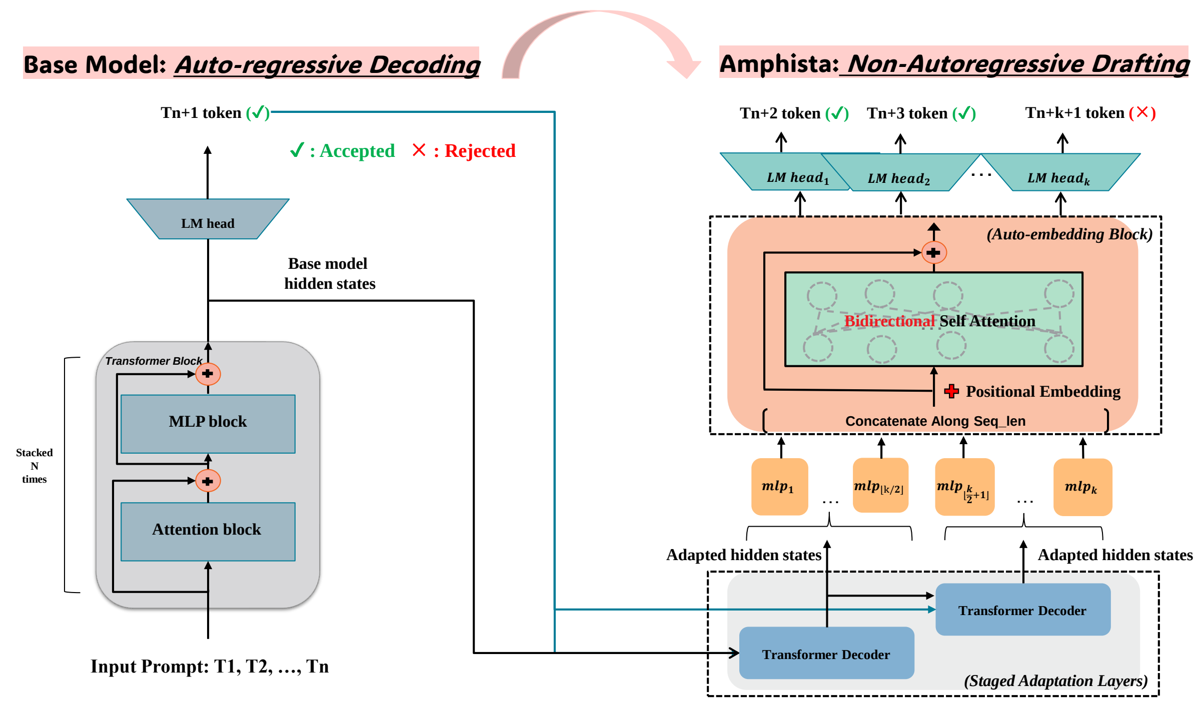

Figure 2: The framework of our Amphista decoding. Our methods improve Medusa in two folds: (1) We introduce staged adaptation layers, consisting of a group of causal Transformer Decoder layers built upon the base LLM, to adapt the base model's hidden states and sampled tokens in two stages. This module ensures that the adapted features contain richer contextual information, supporting multiple-token predictions rather than focusing solely on the immediate next-token prediction. (2) We introduce an auto-embedding block, which is a bi-directional Transformer Encoder module with positional encoding. This block allows each head to attend to others, fostering cooperative predictions and thereby enhancing the speculation accuracy during the drafting stage.

Medusa also uses tree attention, through the use of a specially designed tree causal mask, each node in the tree can only attend to its ancestors, ensuring the accurate computation of attention scores and efficiently processing multiple candidate sequences simultaneously (see A.1 for more details).

3 Amphista

The overview of our method is shown in Figure 2. Building its pipeline upon base model, Amphista contains two main modules: (1) Staged Adaptation Layers. They are causal Transformer Decoder layers that adapt the base model's hidden states and sampled token embedding in two stages, each focusing on different drafting positions. This adaptation process results in hidden states that are enhanced with position-aware contextual information, improving overall prediction accuracy, especially for the latter steps. (2) Auto-embedding Block. It is a Transformer Encoder module that conducts *bi-directional* self-attention computations among the representations of different draft heads, allowing each head can be attended by the others. This facilitates collaborative prediction among these heads, thereby improving overall prediction accuracy.

3.1 Staged Adaptation Layers

Figure 2 demonstrates the relevant details of our staged adaptation layers. Although base model's hidden states contain semantically rich information, there are still differences in the representation requirements between the base model and the draft heads. Specifically, the hidden states of the base model are trained only for predicting the next token, while draft heads need more contextual and positon-aware hidden states to perform multi-step speculation. To address this problem, Medusa-2 applies LoRA (Hu et al., 2021) for joint training of the base model and draft heads, which may compromise the generality on downstream tasks. Hydra employs a single prefix layer for all positions, lacking targeted adaptation for different positions. We propose an effective adaptation method by incorporating two adaptation layers to transform and adapt the strong semantic information from the base model in stages. Specifically, given the hidden states h_t at position t from the base model's final layer and the embedding of the token e_{t+1} sampled from h_t, we use the two adaptation layers to transform them in stages as below:

$$
\begin{aligned}
h_t^1 &= S^1AL(fc_1([h_t; e_{t+1}]), kv_{1:t-1}^1) \\
h_t^2 &= S^2AL(fc_2([h_t^1; e_{t+1}]), kv_{1:t-1}^2)
\end{aligned}
\tag{1}
$$

Note that S^1AL stands for the Stage-one Adaptation Layer that adapts base model hidden states and base token embedding, while S^2AL stands for the Stage-two Adaptation Layer that adapts FAL's output hidden states as well as the base token embedding. The function fc_1 and fc_2 are fully connected layers employed to transform features derived from the concatenation of hidden states and token embeddings. The terms $kv_{1:t-1}^1$ and $kv_{1:t-1}^2$ represent the key-value caches for each adaptation layer. Subsequently, adapted hidden states h_t^1 and h_t^2 are fed into the first and second halves of the drafting heads respectively, ensuring that each adaptation layer focuses on adapting base model's semantic representations in specific future locations.

3.2 Auto-embedding Block

Figure 2 shows the detailed design of our Auto-embedding Block. Given a set of K drafting MLP heads, MLP_k head is tasked with predicting the token in the $(t+k+1)$-th position. Upon acquiring adapted hidden states h_t^1 and h_t^2 from the first and second staged adaptation layers, we first utilize the MLP layers to project them into more position-aware and semantically rich hidden states:

$$
\begin{aligned}
h_k^{'} &= \text{MLP}_k(h_t^1), \qquad k = 1, 2, \ldots, \lfloor K/2 \rfloor \\
h_k^{'} &= \text{MLP}_k(h_t^2), \quad k = \lfloor K/2 \rfloor + 1, \ldots, K
\end{aligned}
\tag{2}
$$

Where $\text{MLP}_i \in \mathbb{R}^{d \times d}$, and d is the dimension of the base model hidden states. We then concatenate these K hidden states along the seq_len dimension:

$$
H' = \text{concat}([h_1', h_2', h_3', \ldots, h_K']) \tag{3}
$$

Where $H' \in \mathbb{R}^{K \times d}$. In order to further enhance the relative positional information among different heads, we introduce additional positional encodings. Specifically, we introduce a learnable positional embedding $PE \in \mathbb{R}^{K \times d}$, and the position-encoded hidden states H_p are expressed as:

$$
H_p = H' + PE \tag{4}
$$

Finally, we employ an effective and efficient bi-directional self-attention module to enable mutual awareness among the drafting heads and use additional learnable lm_head to sample the top-k draft tokens in each position:

$$
attn_o = \text{Self-Attention}(H_p) \tag{5}
$$

$$
draft_k = \text{lm\_head}_k(attn_o[k]), \ k = 1, \ldots, K \tag{6}
$$

In the end, these draft tokens are organized into a draft tree and then verified by the LLM through tree attention. Unlike the independent heads in Medusa and the sequentially dependent heads in Hydra, our Amphista adopts bi-directionally dependent heads. This approach enhances overall prediction accuracy while maintaining a non-autoregressive mechanism, potentially reducing the substantial computation overhead associated with sequential calculations (i.e., autoregressive manner).

3.3 Training Objective

Our loss function consists of two components. The first component aims to match the distribution of the base model's output tokens by employing a Cross-Entropy (CE) loss between the logits of Amphista and those of the base model. The second component uses a language modeling (LM) loss to measure the discrepancy between Amphista's output and the ground truth tokens. This dual objective enables Amphista to align with the base model while also acquiring predictive capabilities from the real corpus to a certain extent.

$$
\mathcal{L}_{\text{Amphista}} = \lambda_1 \mathcal{L}_{\text{alignment}} + \lambda_2 \mathcal{L}_{\text{lm}} \tag{7}
$$

$$
\mathcal{L}_{\text{alignment}} = \text{CE}(logits_{\text{Amphista}}, logits_{T_{t+1}}) \tag{8}
$$

$$
\mathcal{L}_{\text{lm}} = \text{CE}(logits_{\text{Amphista}}, y_{\text{ground\_truth}}) \tag{9}
$$

Note that $logits_{\text{Amphista}}$ and $logits_{T_{t+1}}$ are the logits from Amphista and the base model for token T_{t+1}, while $y_{\text{ground\_truth}}$ represent the ground truth labels of token T_{t+1}. The terms λ_1 and λ_2 are weighting factors for the two training objectives.

4 Experiments

4.1 Experimental Settings

Models and Baselines. Following (Cai et al., 2024; Li et al., 2024; Ankner et al., 2024), we use Vicuna family of models (Zheng et al., 2024) as our base model. Specifically, we implement our method on Vicuna 7, 13, and 33B models with four drafting heads. As for compared baseline methods, we choose original Speculative Decoding, Lookahead

Table 1: The speed-up metric comparison on MT-Bench and Spec-Bench between different methods under **greedy** setting (Temperature = 0). We regard the speed-up of vanilla autoregressive decoding as 1.00×.

Model Size	Method	MT-Bench	Spec-Bench					Avg
			Translation	Summarization	QA	Math	RAG	
7B	Vanilla	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
	Spec-decoding	1.62×	1.11×	1.66×	1.46×	1.45×	1.61×	1.45×
	Lookahead	1.44×	1.15×	1.26×	1.25×	1.56×	1.13×	1.27×
	Medusa	1.87×	1.42×	1.42×	1.50×	1.74×	1.39×	1.50×
	Hydra++	2.37×	1.92×	1.80×	**1.94**×	2.43×	2.04×	2.03×
	Amphista (ours)	**2.44**×	**1.96**×	**2.11**×	**1.94**×	**2.45**×	**2.20**×	**2.13**×
13B	Vanilla	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
	Spec-decoding	1.66×	1.17×	1.75×	1.44×	1.59×	1.73×	1.53×
	Lookahead	1.34×	1.08×	1.23×	1.15×	1.51×	1.15×	1.22×
	Medusa	1.85×	1.55×	1.55×	1.53×	1.88×	1.51×	1.60×
	Hydra++	2.34×	1.75×	1.85×	1.85×	2.31×	1.86×	1.92×
	Amphista (ours)	**2.49**×	**1.88**×	**2.14**×	**1.88**×	**2.41**×	**2.04**×	**2.07**×
33B	Vanilla	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
	Spec-decoding	1.73×	1.28×	1.76×	1.54×	1.71×	1.69×	1.60×
	Lookahead	1.32×	1.09×	1.21×	1.16×	1.55×	1.16×	1.24×
	Medusa	1.97×	1.72×	1.62×	1.66×	2.06×	1.61×	1.73×
	Hydra++	2.54×	1.93×	2.10×	2.04×	2.63×	2.17×	2.17×
	Amphista (ours)	**2.75**×	**2.11**×	**2.49**×	**2.12**×	**2.83**×	**2.44**×	**2.40**×

(Fu et al., 2024), Medusa (Cai et al., 2024) and Hydra (Ankner et al., 2024) for fair comparison.

Training and Datasets. For the training stage, again following (Cai et al., 2024; Ankner et al., 2024), we use ShareGPT [1] dataset to fine-tune our proposed module while keeping base model frozen. Training is conducted using HuggingFace Trainer, which we employ with AdamW optimizer (β_1=0.9, β_2=0.999) and a cosine learning rate schedule with warmup strategy, the initial learning rate is set to 1e-3 and we train 4 epochs. At the evaluation stage, we use MT-Bench (Zheng et al., 2024) and Spec-Bench (Xia et al., 2024) as our benchmark. MT-Bench is an open source multi-turn conversation benchmark which is also evaluated by Hydra and Medusa. Spec-Bench is a well-acknowledged and comprehensive benchmark designed for assessing speculative decoding methods across diverse application scenarios, it includes 480 test samples, encompassing various tasks such as translation, question answering, math reasoning, summarization, and retrieval-augmented generation (RAG).

Metrics. Following previous speculative decoding work, we choose tokens/s and tokens/step as our main metrics. Tokens/step measures the average token length accepted per forward pass of the target LLM. Tokens/s represents the overall throughput of the acceleration algorithm, which is influenced by both the prediction accuracy of the speculator and the drafting latency of the speculator.

[1]ShareGPT. 2023. https://huggingface.co/datasets/Aeala/ShareGPT_Vicuna_unfiltered

4.2 Evaluation of Amphista

Amphista is based on multi-head prediction rather than feature auto-regression prediction. Hence, Hydra, which employs multiple heads for autoregressive drafting, has been chosen as a competitive baseline method for comparison. Specifically, Hydra's best-performing model (i.e., Hydra++) is used for fair evaluation and vicuna-68m (Yang et al., 2024) is used as draft model for the vanilla speculative decoding method. We conduct all the experiments on A100 40G GPUs, and all the experimental settings are kept the same for fair comparison.

Table 1 and Table 2 present the speed-up metrics compared on MT-Bench and Spec-Bench under greedy and random sampling settings (see A.2 for more experiment results). Overall, Amphista demonstrates significant performance superiority over Medusa and surpasses Hydra's best results by a considerable margin across a variety of generation tasks, and also greatly exceeding the speed-up achieved by vanilla speculative decoding. In detail, Amphista achieves a **2.44**× - **2.75**× speed-up on MT-Bench and **2.13**× - **2.40**× speed-up on Spec-Bench under greedy decoding setting. Similarly, under random sampling setting, Amphista achieves a **2.37**× - **2.85**× speed-up and **1.99**× - **2.43**× speed-up on MT-Bench and Spec-

Table 2: The speed-up metric comparison on MT-Bench and Spec-bench between different methods under **random sampling** setting (Temperature = 0.7). We regard the speed-up of vanilla autoregressive decoding as 1.00×.

Model Size	Method	MT-Bench	Spec-Bench					Avg
			Translation	Summarization	QA	Math	RAG	
7B	Vanilla	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
	Spec-decoding	1.39×	1.02×	1.41×	1.24×	1.32×	1.43×	1.28×
	Lookahead	1.28×	1.05×	1.21×	1.12×	1.25×	1.14×	1.16×
	Medusa	1.86×	1.51×	1.47×	1.57×	1.89×	1.43×	1.57×
	Hydra++	2.35×	**1.81**×	1.81×	**1.97**×	2.41×	1.74×	1.95×
	Amphista (ours)	**2.37x**	**1.81**×	**1.92**×	1.96×	**2.43**×	**1.79**×	**1.99**×
13B	Vanilla	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
	Spec-decoding	1.52×	1.08×	1.57×	1.33×	1.42×	1.46×	1.37×
	Lookahead	1.30×	1.07×	1.19×	1.15×	1.38×	1.14×	1.19×
	Medusa	2.01×	1.65×	1.62×	1.71×	2.01×	1.57×	1.71×
	Hydra++	2.57×	1.90×	1.99×	2.12×	2.56×	2.04×	2.12×
	Amphista (ours)	**2.65**×	**1.93**×	**2.16**×	**2.17**×	**2.64**×	**2.15**×	**2.22**×
33B	Vanilla	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
	Spec-decoding	1.58×	1.21×	1.62×	1.48×	1.59×	1.54×	1.48×
	Lookahead	1.29×	1.04×	1.18×	1.15×	1.52×	1.14×	1.21×
	Medusa	2.06×	1.71×	1.79×	1.76×	2.10×	1.79×	1.83×
	Hydra++	2.74×	2.01×	2.24×	2.24×	2.82×	2.26×	2.31×
	Amphista (ours)	**2.85**×	**2.05**×	**2.51**×	**2.29**×	**2.90**×	**2.39**×	**2.43**×

Bench with different base model sizes respectively. These robust results demonstrate that enhancing non-autoregressive drafting can surpass autoregressive drafting in terms of speed-up, highlighting the efficiency of our Amphista architecture. During the drafting stage, all computations in non-autoregressive modeling (i.e., Amphista) can be processed in parallel, better leveraging the parallel computing capabilities of modern GPU accelerators. This leads to a more optimal trade-off between drafting acceptance rate and drafting latency.

Moreover, Amphista exhibits a discernible upward trend in speed-up when employed on larger base models. This can be attributed to Amphista's cost-efficient non-autoregressive modeling and effective transformation of semantic information from the base model. Amphista allows for appropriate increases in accepted token length without introducing excessive additional inference costs. For more exploration on the performance potential of Amphista, please refer to A.2.3.

Last but not least, we further provide the actual throughput of different methods on MT-Bench with a batch size of 1. As depicted in Figure 3, Amphista achieves an actual throughput of approximately 120 tokens/s with a 7B base model and about 80 tokens/s with a 13B base model under both temperature settings. This performance surpasses that of Medusa and Hydra, underscoring Amphista's advantages in practical deployment.

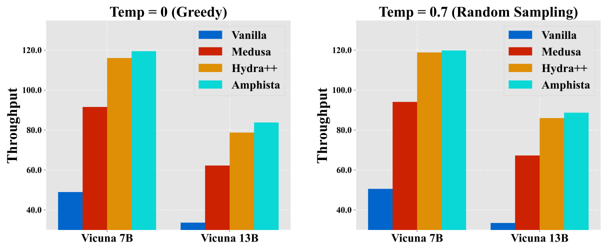

Figure 3: Throughput (tokens/s) on MT-Bench with different base model sizes and temperatures.

Table 3: Results on CNN/DM and XSUM with different temperatures, AR means Auto-Regressive decoding.

Benchmark	Temp	Method	ROUGE-1	ROUGE-2	ROUGE-L	Speed-up
CNN/DM	0.0	AR	18.74	8.44	12.59	1.00×
		Amphista	18.70	8.44	12.59	2.15×
	0.7	AR	17.92	7.65	11.93	1.00×
		Amphista	17.91	7.65	11.92	2.31×
XSUM	0.0	AR	17.32	5.05	12.16	1.00×
		Amphista	17.30	5.05	12.15	2.25×
	0.7	AR	15.99	4.44	11.42	1.00×
		Amphista	15.96	4.43	11.40	2.10×

4.3 Generation Quality of Amphista

We perform evaluation on XSUM (Narayan et al., 2018) and CNN/DM (See et al., 2017) to validate the generation quality of our Amphista (more details can be found in appendix A.2.1). From the ROUGE-1/2/L scores (Lin, 2004) in Table 3, we can find that Amphista can reserve the output distribution quality while achieving 2.10×-2.31× speed-up compared with vanilla auto-regressive decoding.

Table 4: Ablation experiments of different model variants on MT-Bench and Spec-Bench, with the base model being Vicuna 7B and the evaluation metric being **speed-up**. Medusa can be considered as Amphista w/o any added modules, and Hydra can be seen as Medusa w/ sequential dependency heads.

Method Variants	MT-Bench	Spec-Bench					Avg
		Translation	Summary	QA	Math	RAG	
Medusa	1.86×	1.51×	1.47×	1.57×	1.89×	1.43×	1.57×
Hydra++	2.37×	1.92×	1.80×	1.94×	2.43×	2.04×	2.03×
Amphista w/o Auto-embedding	2.30×	1.82×	2.00×	1.81×	2.25×	1.99×	1.97×
Amphista w/o Position-Encoding	2.42×	1.96×	2.08×	1.92×	2.42×	2.18×	2.11×
Amphista w/o Staged-Adaptation	2.14×	1.85×	1.75×	1.78×	2.10×	1.91×	1.88×
Amphista w/ One-Adaptation-Layer	2.31×	1.90×	1.99×	1.83×	2.35×	2.14×	2.04×
Amphista w/o Sampled-Token	2.25×	1.88×	1.80×	1.81×	2.26×	2.01×	1.95×
Amphista (ours)	2.44×	1.96×	2.11×	1.94×	2.45×	2.20×	**2.13**×

Table 5: Ablation experiments of different model variants on MT-Bench and Spec-Bench, with the base model being Vicuna 7B and evaluation metric being **average accepted length**. Medusa can be considered as Amphista w/o any added modules, and Hydra can be seen as Medusa w/ sequential dependency heads.

Method Variants	MT-Bench	Spec-Bench					Avg
		Translation	Summary	QA	Math	RAG	
Medusa	2.52	2.12	2.01	2.05	2.48	2.09	2.15
Hydra++	3.58	2.80	2.70	2.91	3.61	2.90	**2.98**
Amphista w/o Auto-embedding	3.16	2.41	2.66	2.40	3.11	2.49	2.60
Amphista w/o Position-Encoding	3.47	2.61	2.90	2.78	3.47	2.91	2.93
Amphista w/o Staged-Adaptation	2.91	2.42	2.24	2.30	2.85	2.38	2.43
Amphista w/ One-Adaptation-Layer	3.36	2.49	2.68	2.71	3.37	2.75	2.80
Amphista w/o Sampled-Token	3.11	2.43	2.48	2.45	3.15	2.55	2.61
Amphista (ours)	3.50	2.62	3.01	2.80	3.50	2.96	**2.98**

4.4 Ablation Study

Diverging from other approaches based on speculative sampling and Medusa, Amphista's main insight lies in adapting transformation through Staged Adaptation Layers and enhancing integration via the non-autoregressive Auto-embedding Block. This approach strengthens semantic information derived from the base model. In doing so, Amphista achieves significant improvements in drafting accuracy while also maintaining highly efficient parallel computing capabilities. The former experimental results show that Amphista indeed achieves a significant improvement in both drafting accuracy and drafting efficiency. In this section, we conduct comprehensive ablation experiments based on the vicuna 7B model to validate the effectiveness of each proposed module in our Amphista. Specifically, we conduct five model variants as follows: (1) **Amphista w/o Auto-embedding** which means removing the Auto-embedding Block in Amphista. (2) **Amphista w/o Position-Encoding** which means removing the additional position embedding matrix in Auto-embedding Blcok. (3) **Amphista w/o Staged-Adaptation** which means removing staged adaptation layers. (4) **Amphista w/ One-Adaptation-Layer** which means using only one adaptation layer for all the drafting heads. (5) **Amphista w/o Sampled-Token** which means removing sampled token during adaptation process. It should be noted that we consider the original Medusa as Amphista without any additional modules, and we regard Hydra as Medusa with sequentially dependent heads. The corresponding experimental results are presented in Table 4 and Table 5. From these comparative results, four key observations can be found as follows:

- **Amphista w/o Auto-embedding** exhibits an approximate 5%-8% decrease in speed-up performance and about a 10%-12% reduction in average accepted length. This highlights the effectiveness of the Auto-embedding Block in mitigating inaccuracies deriving from the independent speculation of Medusa heads, and demonstrating the efficiency of non-autoregressive drafting computations. Additionally, **Amphista w/o Position-Encoding** exhibits a slight performance decline, with an approximate 2% decrease in inference speed-up, suggesting that position encoding provides additional benefits.

- **Amphista w/o Staged-Adaptation** leads to a more significant decline in speed-up (14%) and average accepted length (16%). This emphasizes the importance of bridging the feature gap between the base model and drafting heads, and further underscores the critical role of the staged adaptation layer in enhancing the auto-embedding block. Additionally, it is noteworthy that **Amphista w/ One-Adaptation-Layer** utilizes only a single adaptation layer for all drafting positions. In contrast to staged adaptation, this approach poses greater challenges to the adaptation process, resulting in some performance degradation, thereby validating the rationale behind our staged adaptation design.

- **Amphista w/o Sampled-Token** also causes an approximate 8% performance decline. Unlike previous works (e.g., Hydra and EAGLE), we do not use the sampled token directly for the next step of prediction. Instead, we adapt it along with the base model's hidden states. This not only indicates that the sampled token, in addition to base model hidden states, contains important semantic information, but also demonstrates the effectiveness of our staged adaptation approach.

- Thanks to the autoregressive characteristics and the substantial number of parameters in the MLP layers, Hydra exhibits great performance in average token length. However, the computational overhead of auto-regressive methods is huge, resulting in significant reductions when translated into final speed-up. In contrast, Amphista achieves a comparable average token length to Hydra, and due to the parallelism and efficiency of its non-autoregressive computations, it ultimately attains a more favorable overall trade-off.

5 Related Work

Increasing techniques have been proposed to enhance the inference speed of large language models (LLMs), covering aspects of system hardware, model architecture, and decoding algorithms. A significant branch of these techniques is **Model Compression**, which includes methods such as model quantization (Yao et al., 2023; Dettmers et al., 2024; Liu et al., 2023a; Ma et al., 2024), pruning (Belcak and Wattenhofer, 2023; Liu et al., 2023b; Zhong et al., 2024), and distillation (Zhou et al., 2024; Sun et al., 2024; Touvron et al., 2021). Additionally, techniques like kv-cache (Ge et al., 2023; Kwon et al., 2023), flash-attention (Dao et al., 2022), and early exiting (Bae et al., 2023; Elhoushi et al., 2024; Liu et al., 2024a) have also significantly reduced inference overhead.

Another important line of research is **Speculative Decoding**, which our work is based on. It can be broadly categorized into two types. The first treats the target model and draft model separately and independently, involving the use of a small language model (Kim et al., 2024; Leviathan et al., 2023; Liu et al., 2024b; Monea et al., 2023; Chen et al., 2024; Du et al., 2024), external database, or n-grams pool (He et al., 2024; Fu et al., 2024; Kou et al., 2024; Ou et al., 2024) to generate candidate token sequences or token trees (Miao et al., 2024), which the LLM then verifies. The second type views the draft model as a dependent approximation of the target model, deriving the draft model directly from the target model or building additional modules on top of the base model for drafting. For instance, Self-SD (Zhang et al., 2023) utilizes the LLM itself by skipping some decoder layers for drafting, ReDrafter (Zhang et al., 2024) uses an RNN-style structure to generate draft tokens, and EAGLE (Li et al., 2024) trains a feature regressive layer to predict subsequent tokens. Medusa (Cai et al., 2024), Clover (Xiao et al., 2024), and Hydra (Ankner et al., 2024), which are most similar to our work, use lightweight drafting heads to obtain candidate token trees. Unlike these approaches, we propose a novel method using a bi-directional auto-embedding block combined with additional staged adaptation layers to further enhance acceleration.

6 Conclusion

We propose Amphista, an efficient non-autoregressive speculative decoding framework that accelerates inference through parallel processing and enhances alignment between the base and draft models via feature adaptation. Specifically, Amphista introduces two key modules: the Auto-embedding Block, which uses bi-directional self-attention to enable collaborative speculation among drafting heads, and the Staged Adaptation Layers, which transform semantic information from the base model for multi-step prediction. Extensive experiments demonstrate the effectiveness and superiority of Amphista, showcasing the potential of non-autoregressive modeling for speculative decoding.

Limitations

While we have found and adhered to using bidirectional self-attention for non-autoregressive modeling as an efficient inference structure, we have not yet fully explored the optimal structure of the Auto-embedding Block module. Specifically, this includes experimenting with different intermediate sizes (i.e., the hidden dimensions used in self-attention computations) and increasing the number of self-attention layers within the auto-embedding block to enhance its modeling depth (see A.2.3). Both of these structural optimizations could potentially improve Amphista's acceleration performance within the current framework. Additionally, this work primarily focuses on scenarios where the batch size is equal to one, with limited consideration and optimization for larger batch sizes. We leave these areas as our future work and also hope that researchers interested in non-autoregressive inference acceleration will build upon this foundation.

References

Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. 2024. Hydra: Sequentially-dependent draft heads for medusa decoding. *Preprint*, arXiv:2402.05109.

Sangmin Bae, Jongwoo Ko, Hwanjun Song, and Se-Young Yun. 2023. Fast and robust early-exiting framework for autoregressive language models with synchronized parallel decoding. pages 5910–5924, Singapore.

Peter Belcak and Roger Wattenhofer. 2023. Exponentially faster language modelling. *arXiv preprint arXiv:2311.10770*.

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. *Preprint*, arXiv:2401.10774.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating large language model decoding with speculative sampling. *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, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. 2024. Sequoia: Scalable, robust, and hardware-aware speculative decoding. *Preprint*, arXiv:2402.12374.

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

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Preprint*, arXiv:2205.14135.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36.

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

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed A Aly, Beidi Chen, and Carole-Jean Wu. 2024. Layerskip: Enabling early exit inference and self-speculative decoding. *Preprint*, arXiv:2404.16710.

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

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2023. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D. Lee, and Di He. 2024. Rest: Retrieval-based speculative decoding. *Preprint*, arXiv:2311.08252.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2021. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Sehoon Kim, Karttikeya Mangalam, Suhong Moon, Jitendra Malik, Michael W Mahoney, Amir Gholami, and Kurt Keutzer. 2024. Speculative decoding with big little decoder. *Advances in Neural Information Processing Systems*, 36.

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

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626.

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. *Preprint*, arXiv:2401.15077.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Kai Han, and Yunhe Wang. 2024a. Kangaroo: Lossless self-speculative decoding via double early exiting. *Preprint*, arXiv:2404.18911.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. 2024b. Online speculative decoding. *Preprint*, arXiv:2310.07177.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. 2023a. Llm-qat: Data-free quantization aware training for large language models. *Preprint*, arXiv:2305.17888.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. 2023b. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pages 22137–22176. PMLR.

Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Li Dong, Ruiping Wang, Jilong Xue, and Furu Wei. 2024. The era of 1-bit llms: All large language models are in 1.58 bits. *Preprint*, arXiv:2402.17764.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. 2024. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 932–949.

Giovanni Monea, Armand Joulin, and Edouard Grave. 2023. Pass: Parallel speculative sampling. *arXiv preprint arXiv:2311.13581*.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *ArXiv*, abs/1808.08745.

OpenAI. 2022. Chatgpt: Chatgpt: Optimizing language models for dialogue.

Jie Ou, Yueming Chen, and Wenhong Tian. 2024. Lossless acceleration of large language model via adaptive n-gram parallel decoding. *arXiv preprint arXiv:2404.08698*.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

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

Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. 2024. Spectr: Fast speculative decoding via optimal transport. *Advances in Neural Information Processing Systems*, 36.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. 2021. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pages 10347–10357. PMLR.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

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. *Preprint*, arXiv:2401.07851.

Bin Xiao, Chunan Shi, Xiaonan Nie, Fan Yang, Xiangwei Deng, Lei Su, Weipeng Chen, and Bin Cui. 2024. Clover: Regressive lightweight speculative decoding with sequential knowledge. *arXiv preprint arXiv:2405.00263*.

Sen Yang, Shujian Huang, Xinyu Dai, and Jiajun Chen. 2024. Multi-candidate speculative decoding. *Preprint*, arXiv:2401.06706.

Zhewei Yao, Cheng Li, Xiaoxia Wu, Stephen Youn, and Yuxiong He. 2023. A comprehensive study on post-training quantization for large language models. *arXiv preprint arXiv:2303.08302*.

Aonan Zhang, Chong Wang, Yi Wang, Xuanyu Zhang, and Yunfei Cheng. 2024. Recurrent drafter for fast speculative decoding in large language models. *Preprint*, arXiv:2403.09919.

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

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

Shuzhang Zhong, Zebin Yang, Meng Li, Ruihao Gong, Runsheng Wang, and Ru Huang. 2024. Propd: Dynamic token tree pruning and generation for llm parallel decoding. *arXiv preprint arXiv:2402.13485*.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. 2024. Distillspec: Improving speculative decoding via knowledge distillation. *Preprint*, arXiv:2310.08461.

A Appendix

A.1 Draft Tree

For a fully fair comparison, we adopt the same draft tree structure as Medusa and Hydra. As shown in Figure 4, this tree is a sparse structure with a depth of 4, representing four drafting heads, and includes a total of 64 nodes, including the root node (the token sampled in the final step of the base model). Each layer's nodes represent the tokens obtained by top_k sampling from the corresponding drafting head. The entire tree is constructed using an auxiliary dataset by maximizing the acceptance probability of the whole tree (Cai et al., 2024). Moreover, a specially designed tree mask is used to correctly compute attention scores while simultaneously handling multiple paths, as described in Figure 5.

However, in some cases, due to the lack of redundant computational power (such as in high-throughput inference service scenarios) or parallel accelerators, an excessive number of tree nodes may lead to significant computation overhead, thereby affecting the acceleration efficiency of the algorithm. Consequently, we configure varying numbers of draft tree nodes without changing the tree depth for more comprehensive comparison, and the experimental results are shown in Table 6. From these results we observe that as the number of tree nodes decreases, the width of the tree reduces, leading to a decrease in speed-up for all compared methods. However, the decline is slightly less pronounced for Amphista, owing to its higher head accuracy. Furthermore, across various tree node configurations, we consistently achieve optimal performance, demonstrating the advantages of our algorithm in practical deployment and low-resource scenarios.

Table 6: Speed-up comparison on MT-Bench with varying number of draft tree nodes.

Method	Node = 22	Node = 35	Node = 45	Node = 64
Medusa	1.71×	1.80×	1.87×	1.87×
Hydra++	2.17×	2.26×	2.28×	2.37×
Amphista	**2.29**×	**2.37**×	**2.42**×	**2.44**×

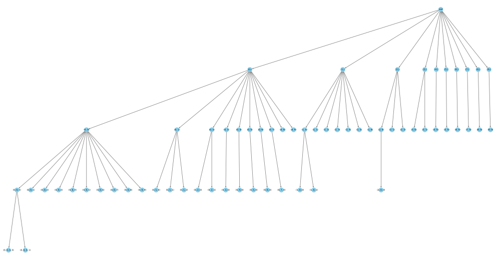

Figure 4: Draft tree used in Medusa, Hydra and our Amphista.

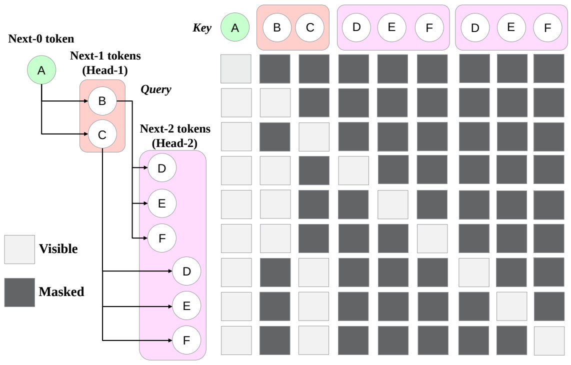

Figure 5: An illustration of Tree Attention. Assuming Medusa has only 2 heads, where head-1 generates the top-2 tokens and head-2 generates the top-3 tokens, resulting in 6 candidate sequences (e.g., ABD). Additionally, a special tree mask is designed to ensure causal relationships among the top-k nodes of each head.

A.2 Additional Experiments Results

A.2.1 Evaluation on XSUM and CNN/DM

We use XSUM (Narayan et al., 2018) and CNN/DM (See et al., 2017) for evaluating the generation quality of our Amphista, the base model is vicuna 7B. Specifically, we perform zero-shot evaluation and the input prompt template is `'Article:'+ 'Original Text' + '\nSummary:'`. Additionally, for input prompts exceeding a length of 2048, we perform truncation to meet the base model's input requirements.

Table 7: The speed-up metric comparison on Humaneval and GSM8K between different methods under greedy setting. The base model is vicuna 7B and 13B, and we regard the speed-up of vanilla auto-regressive decoding as 1.00×.

Model Size	Benchmark	Vinilla AR	Medusa	Hydra++	Amphista
7B	Humaneval	1.00×	2.40×	2.76×	**3.02**×
	GSM8K	1.00×	1.87×	2.14×	**2.32**×
13B	Humaneval	1.00×	2.11×	2.75×	**3.00**×
	GSM8K	1.00×	1.98×	2.39×	**2.68**×

A.2.2 Code Generation and Math Reasoning

In this section, we provide more experimental results on code generation and math reasoning. we choose public Humaneval (Chen et al., 2021) and GSM8k (Cobbe et al., 2021) benchmark for evalu-

Table 8: The speed-up and average accepted length metric comparison with the base model being vicuna 7B. We regard the speed-up of vanilla auto-regressive decoding as 1.00×.

Metric	Method	MT-Bench	Spec-Bench					Avg
			Translation	Summarization	QA	Math	RAG	
Speed-up	Vanilla	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
	Hydra++	2.37×	1.92×	1.80×	1.94×	2.43×	2.04×	2.03×
	Amphista	2.44×	1.96×	2.11×	1.94×	2.45×	2.20×	2.13×
	Amphista-α	**2.63**×	**2.09**×	**2.23**×	**2.06**×	**2.61**×	**2.34**×	**2.27**×
Average Accepted Length	Vanilla	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Hydra++	3.58	2.80	2.70	2.91	3.61	2.90	2.98
	Amphista	3.50	2.62	3.01	2.80	3.50	2.96	2.98
	Amphista-α	3.58	2.70	3.14	2.90	3.62	3.08	**3.09**

ation, and the base model is vicuna 7B and vicuna 13B. According to the results in Table 7, we can observe that due to the universal template and notation of code generation and mathematical reasoning, almost all compared methods achieve a higher speed-up. Furthermore, our Amphista algorithm consistently attains optimal performance, demonstrating the superiority of our approach.

A.2.3 Exploring The Potential of Amphista

In this section, we conduct a preliminary exploration of Amphista's scaling ability to demonstrate its potential for performance enhancement. By leveraging the efficiency of non-autoregressive modeling, we increase the number of auto-embedding blocks, which are essential modules within Amphista, while maintaining parallel inference. This approach yields remarkable results, detailed in Table 8. Specifically, we employ two layers of self-attention in the auto-embedding module, renaming our method as Amphista-α. This adjustment leads to an average accepted length increase of approximately 0.1-0.2 tokens and a notable 5%-8% improvement in overall speed-up, highlighting Amphista's performance growth potential. We anticipate this to be a highly promising and potent attribute of Amphista.

A.3 Case Study

Here we show some real case studies (see Figure 6, 7) on Amphista inference, the base model is Vicuna 7B. Note that we do not apply any special processing to the tokenizer's output, preserving the original results. Tokens highlighted in **red** represent those generated by our Amphista during each step of decoding. Tokens in **black** indicate those generated by base model. From these practical examples, we can observe that in the vast majority of cases, Amphista generates at least two tokens

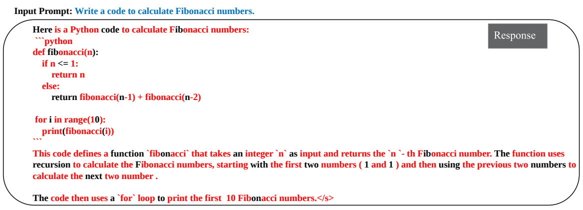

Figure 6: Case study on code generation. Tokens in **red** means those generated by our Amphista and tokens in **black** means those generated by base model itself.

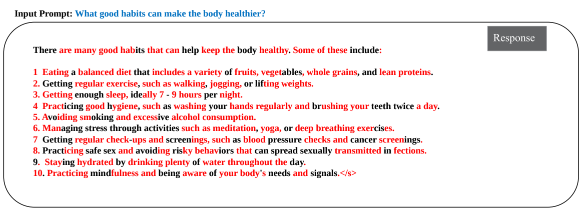

Figure 7: Case study on text generation. Tokens in **red** means those generated by our Amphista and tokens in **black** means those generated by base model itself.

per decoding step. This generally results in a stable at least 2x speed-up, demonstrating the efficiency of our algorithm. Additionally, Amphista's output is consistent with the base model's auto-regressive decoding output, ensuring the generation quality of our Amphista.