Parallel Decoding via Hidden Transfer for Lossless Large Language Model Acceleration

Anonymous ACL submission

Abstract

Large language models (LLMs) have recently shown remarkable performance across a wide range of tasks. However, the substantial number of parameters in LLMs contributes to significant latency during model inference. This is particularly evident when utilizing autoregressive decoding methods, which generate one token in a single forward process, thereby not fully capitalizing on the parallel computing capabilities of GPUs. In this paper, we propose a novel parallel decoding approach, namely *hidden transfer*, which decodes multiple successive tokens simultaneously in a single forward pass. The idea is to transfer the intermediate hidden states of the previous context to the *pseudo* hidden states of the future tokens to be generated, and then the pseudo hidden states will pass the following transformer layers thereby assimilating more semantic information and achieving superior predictive accuracy of the future tokens.

Besides, we use the novel tree attention mechanism to simultaneously generate and verify multiple candidates of output sequences, which ensure the lossless generation and further improves the generation efficiency of our method. Experiments demonstrate the effectiveness of our method. We conduct a lot of analytic experiments to prove our motivation. In terms of acceleration metrics, we outperform all the single-model acceleration techniques, including Medusa and Self-Speculative decoding.

1 Introduction

Recent developments in Transformer-based large language models (Vaswani et al., 2017; Radford et al., 2018, 2019; Brown et al., 2020; Zeng et al., 2022; Zhang et al., 2022; Touvron et al., 2023a,b; Roziere et al., 2023) have demonstrated remarkable performance across a broad spectrum of tasks. However, these models grapple with excessive inference latency due to the inherently serial process of generating one token per forward pass, the inference process is typically memory bandwidth-bound, which means most of the time model inference is spent loading billion parameters from memory rather than computing, resulting in the waste of the parallel computational power of GPUs (Shazeer, 2019). In our experiments, We find that due to the parallelism of GPU computing, the time for multiple tokens to propagate in parallel is nearly the same as the time for one token to propagate(as shown in Table 1). Therefore, the low efficiency of the inference stage becomes the biggest bottleneck to broaden the application scenarios of LLMs.

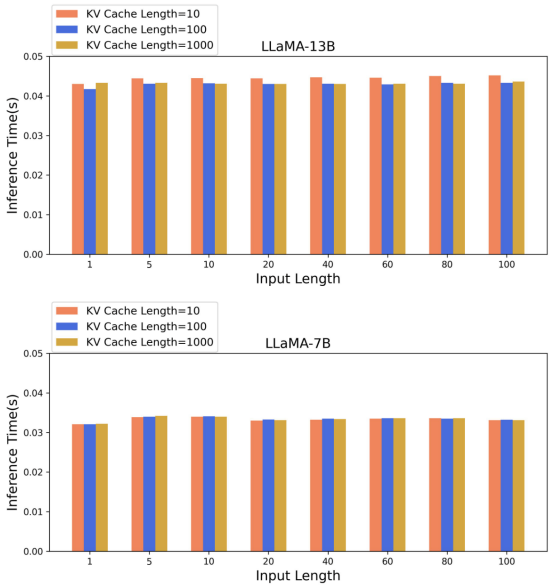

Figure 1: A single forward time consumption for various LLMs with different size under different KV cache length and different number of input tokens, each data is the average of randomly 100 samples. The result shows that under the setting of KV cache, increasing the length of input sequence in a certain range will not increase the time of forward propagation, and then prove that the traditional autoregressive decoding has a waste in GPU utilization efficiency

To address this bottleneck, contemporary work

proposes speculative decoding (Leviathan et al., 2023; Chen et al., 2023a; Zhang et al., 2023), which utilizes a small language model to draft a few tokens ahead. then the LLM verifies the drafted tokens and accepts the correct ones. While this acceleration technique achieves promising performance, it has its limitations: speculative decoding requires another model to do the draft thing. It is inconvenient to cooperate with an extra model in some scenarios as it requires more sophisticated scheduling and the draft model might consume extra GPU and memory resources. Thus, some researchers have been investigating single-model acceleration. That is, to speed up the inference of LLMs without auxiliary models. Self-speculative decoding (Zhang et al., 2023) and Medusa (Cai et al., 2023) are typical methods within this line of research. Medusa predicts not only the next token within a single forward propagation but also a few tokens ahead of the next token. These extra tokens are predicted based on the last hidden states of the input tokens through the trainable Medusa heads.

Our method aligns with the line of single-model acceleration. We design a novel method called **Hidden Transfer**, to predict the *pseudo* hidden states of future tokens in the intermediate layers. We use a trainable linear projection to transfer the hidden states of input tokens to the pseudo hidden states of future tokens in a certain intermediate layer, and the synthesized pseudo hidden states pass the subsequent layers and interact with hidden states of the whole sequence as normal, in the last layer, we use the original lm-head and decode the draft tokens of the future positions. In this way, we can predict more than merely the next token but also a few tokens ahead in a single forward propagation. In the training stage we employ KL-divergence as the supervised signal, which minimizes the distribution between tokens predicted by the pseudo hidden states and the real ones. In addition to the novel design of Hidden Transfer, we also use the tree attention mechanism (Cai et al., 2023; Miao et al., 2023; Spector and Re, 2023) to simultaneously perform the token prediction and token verification to ensure the lossless generation of our method.

The motivation for hidden transfer is that the synthesized pseudo hidden states will interact with themselves and previous hidden states of the context during the forward propagation in which gain more semantic information to boost the success rate of predicting future tokens. Our experiments show that this motivation is fulfilled, and our method can achieve the best draft token prediction accuracy and inference acceleration ratio compared with other methods under a single-model setting.

Our key contributions are: (1) To our best knowledge, we are the first to study the prediction of pseudo hidden states of the future tokens in LLMs, our experiments prove that intermediate hidden states could be predicted directly and refined in the forward propagation. (2) We propose Hidden Transfer, a novel single-model lossless acceleration method for improving the inference efficiency of LLMs. Our method predicts multiple draft tokens with synthetic pseudo hidden states. (3) We conduct various experiments to prove the effectiveness of our method, including some analytic experiments to prove our motivation.

2 Related Work

To solve the problem of inference latency in LLMs, the existing works can be divided into the following two categories: we call the first category **Model Compression**, including model distillation (Sanh et al., 2019), model pruning (Frantar and Alistarh, 2023; Wang et al., 2021) and model quantization (Liu et al., 2023a), aiming to replace the original large language model with a small model which have the similar but not identical outputs in certain field. We refer the second category of methods as **Speculative Decoding**, the key idea of is to reduce the number of forward propagation of the LLMs under the condition that the generated results remain unchanged.

2.1 Model Compression

There are plenty of works focus on the model lightweight, including model quantization (Han et al., 2015; Zhao et al., 2019; Jacob et al., 2018; Yao et al., 2022; Frantar et al., 2022; Liu et al., 2023a), knowledge distillation (Hinton et al., 2015; Cho and Hariharan, 2019; Hsieh et al., 2023), model pruning (Xia et al., 2023; Guo et al., 2023; Chen et al., 2023b) and model sparsification (Hoefler et al., 2021; Liu et al., 2023b). Model quantization is to convert the model parameters to floating-point numbers with lower precision or integers; Model pruning and sparsification removes redundant components in the LLMs; Knowledge distillation works by transferring knowledge from a teacher model to a student model (small model). These model compression methods have a wide range of applications, but they do not guarantee

that the output is strictly consistent with the original LLMs.

2.2 Speculative Decoding

This series of methods aim to quickly generate some draft tokens and use the LLMs to verify them in parallel to keep lossless generation theoretically. We can divide this class of methods into two types based on the number of deployed models, single model and multiple models. The single models' approach is represented by Medusa (Cai et al., 2023), and Self-speculative decoding method (Zhang et al., 2023), Medusa and train extra multiple heads to predict subsequent tokens based on the last hidden state of the input tokens after a single forward propagation. Self-speculative methods use a subset of intermediate layers of the whole LLM as the draft model to generate draft tokens The multiple models' approach is represented by traditional speculative decoding (Leviathan et al., 2023; Chen et al., 2023a), which uses a small language models (SLMs) as the draft model to generate draft tokens, the LLMs verify the tokens in parallel.

3 Methodology

In this section, we first define the problem formulation, including the overview of traditional autoregressive decoding algorithm and parallel decoding algorithm, then we provide a detailed description of the training and testing process of our hidden transfer method.

3.1 Problem Formulation

Transformer-based auto-regressive language models aim to construct the the $(n+1)_{th}$ token's distribution given the prefix n tokens, denoted as $P(x_{n+1}|x_1, x_2, \ldots, x_n)$ These models are capable of processing entire sequences in parallel during the training stage. However, during inference stage, the generation process becomes serial naturally. This is due to the requirement of the preceding n tokens' semantic information to predict the probability distribution of the $(n+1)_{th}$ token. Traditional autoregressive generation techniques, whereby a single token is produced per model forward process, fail to capitalize on the parallel processing capabilities of GPUs, consequently impacting the response efficiency of various AI systems. The essence of parallel decoding algorithms, exemplified by speculative decoding, lies in their ambition to increase the expected number of tokens generated in one single forward propagation of the LLMs while maintain the generation consistency. Thus, the optimization goal of this class of methods can be written to find a maximum positive integer k that satisfies the following conditions:

$$\tilde{P}(X_{n+k+1} \ldots X_{n+1}|X_{\leq n}) = P(X_{n+k+1} \ldots X_{n+1}|X_{\leq n})$$

Where P and $\tilde{P}$ represent the token distribution given by the original language model and parallel decoding algorithm respectively. For most parallel decoding algorithms, $\tilde{P}$ is obtained by a draft and verify process, tree attention is wildly adopted to verify multiple candidate sequences simultaneous, so a lot of works focus on how to generate better candidates in the draft stage, the process can be described as the following formula:

$$X_{n+k+1}, \ldots, X_{n+1} = M(X_{\leq n})$$

In some speculative decoding algorithms, M represents small language models, or part of the original LLMs (Zhang et al., 2023), in the block-wise series of methods (represented by Medusa (Cai et al., 2023)), M can be viewed as the extra heads in the last layer of LLMs, In our method, M can be viewed as the hidden transfer linear projection in some intermediate layers, in the following section, we concisely introduce our method, including the training and inference stages.

3.2 Hidden Transfer

The core idea of our hidden transfer is to predict the future $k+1$ tokens(including k draft tokens and the next token generated correctly by language model) by one step of LLM forward propagation without the deployment of SLMs, existing works under this setting either use earlying exiting method to directly predict the token distribution (Bae et al., 2023) or train extra k lm-heads to predict the k draft tokens in the last layer, we believe that the first method will lose a lot of information at higher layers, assuming that we use X_n to represent the n_{th} token of the input sequence, and h_n^j represents the j_{th} layer's hidden state of the n_{th} token, the first method trains an early lm-head to predict X_{n+1} in advance, but once the X_{n+1} is predicted, it should be used as the input in the next round of forward propagation, the previous forward propagation will stop in the middle layer, as a result the higher layers' hidden state of X_n will be lost(i.e. h_n^{j+1}, h_n^{j+2} $\ldots$), although there're some works claim that they can simply use copy mechanism to simulate the

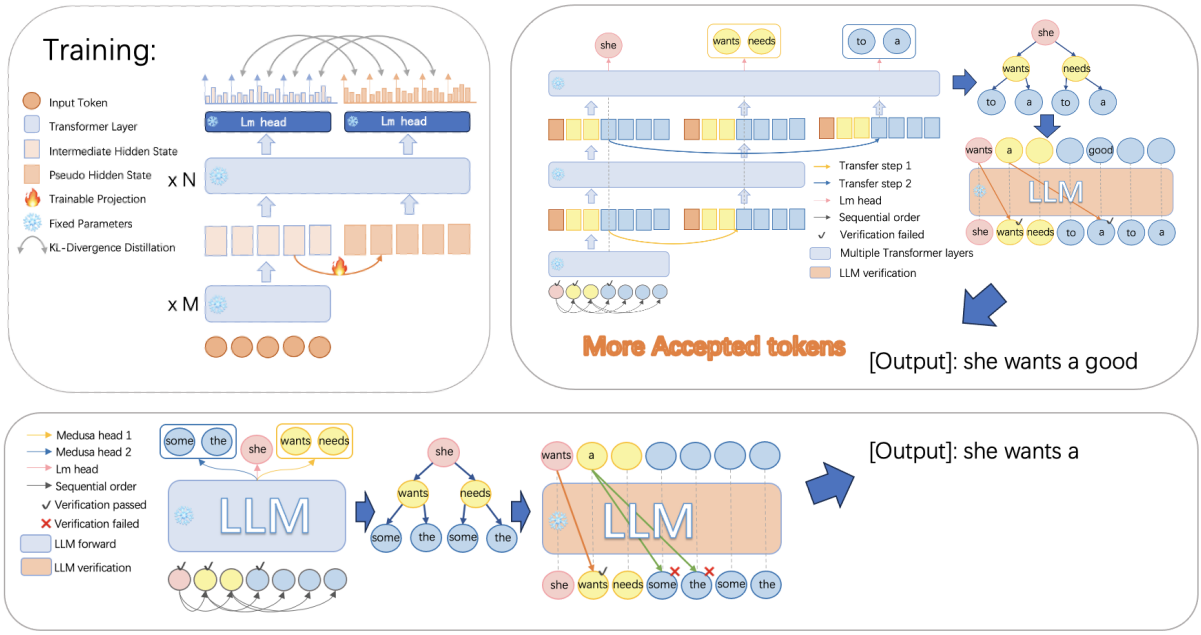

Figure 2: Overview of our method. The upper-left is the training process of hidden transfer, N and M represent the numbers of transformer layers. The upper-right and the bottom of the figure are the inference process of Hidden transfer and Medusa respectively, assuming that the generation of both methods starts from the same context and their inputs are the candidate token sequences generated in the last round, Medusa and Hidden transfer both verify the candidate token sequences to find the last accepted token position(i.e the fourth token of the input) and generate the next token and new draft tokens at the same time(for simplicity we only consider 2 transfer steps/medusa heads), the next token and draft tokens construct to a tree structure and are then flatted into a sequence to be verified with tree attention, after the verification stage, result shows Hidden transfer has more prediction accuracy and more draft tokens accepted

higher layers' hidden state (Elbayad et al., 2019), but other experiments still show that the copy mechanism performs badly in some cases (Bae et al., 2023). The second method which use extra lm-heads to predict the future k draft tokens is very simple and effective, but we believe this method lacks the interaction of the k draft tokens and the previous tokens because the draft tokens do not interact with the previous tokens through the attention mechanism, for example it only use the last hidden states of X_n to predict the X_{n+2} without using the X_{n+1}'s information, but some times the X_{n+2} depends on the X_{n+1}, so our method choose to train multiple transfer functions (simple linear projections) to **predict** the future k draft token's hidden states in some intermediate layers by mapping the h_n^j to h_{n+1}^j ... h_{n+k}^j, so we can continue the forward propagation with $n+k$ intermediate hidden states, During the following transformer layers, the last k hidden states will pass the original lm-head to predict the token distribution normally, the whole training and inference process compared with Hidden transfer and Medusa is shown in Figure2.

3.3 Training stage

At the training stage, it is required to train multiple linear projections in multiple fixed layers, where the locations of the corresponding transfer layers and the number of transfer step are considered as hyperparameters. The number of transfer step is equal to the number of pseudo hidden states predicted in a single forward process for one token; hence, we denote this number as k. Because we train k linear projections separately so we need to conduct the training process k times(we train one linear projection for one transfer step in a certain layer). For simplicity we discuss the training process of the i_{th} transfer step, we denote the index of transfer layer for step i as t_i. so we can use $W_{t_i}^i \in \mathbb{R}^{d\times d}$ $(1 \leq i \leq k)$ to denote the trainable linear projection for the i_{th} step(d represent the hidden dimension of the LLMs). Assume we have original token sequences X_1, X_2, ..., X_n, we first do the forward process to the t_i layer to get their corresponding hidden state $h_1^{t_i}$, $h_2^{t_i}$, ..., $h_n^{t_i}$, then we transfer all of the n hidden states into their

corresponding pseudo hidden states, which can be formulated as below:

$$\widetilde{h}_n^{t_i}, \widetilde{h}_{n-1}^{t_i}, \ldots, \widetilde{h}_1^{t_i} = W_{t_i}^i \cdot (h_n^{t_i}, h_{n-1}^{t_i}, ..., h_1^{t_i})$$

After transferring the $h_j^{t_i}$ into $\widetilde{h}_j^{t_i}$ ($1 \leq j \leq n$), we concat the original hidden states and the pseudo hidden states into a new sequence (ie.$h_1^{t_i}, ..., h_{n-1}^{t_i}, h_n^{t_i}, \widetilde{h}_1^{t_i}, \ldots, \widetilde{h}_{n-1}^{t_i}, \widetilde{h}_n^{t_i}$). It's easy to show that the $\widetilde{h}_j^{t_i}$ is the pseudo hidden state of h_{j+1}^i, so in the self-attention layers it can only view the hidden states from $h_1^{t_i}$ to $h_{j+i-1}^{t_i}$ and itself in the sequence, we design attention mask to achieve the goal. In order to ensure the consistency between the training stage and inference stage, we set the position id $j+i$ to $\widetilde{h}_j^{t_i}$ to construct the position embedding.

We then continue to forward the new sequence, and get the final representations of the last layer (i.e., $h_1^l, \ldots, h_{n-1}^l, h_n^l, \widetilde{h}_1^l, \ldots, \widetilde{h}_{n-1}^l, \widetilde{h}_n^l$, where l denotes the number of layers of the LLMs). We then pass the hidden states sequence through the original lm-head and finally get the token distribution of each position ($P_1^l, \ldots, P_{n-1}^l, P_n^l, \widetilde{P}_1^l, \ldots, \widetilde{P}_{n-1}^l, \widetilde{P}_n^l$). We use the KL-divergence between token distributions given by the pseudo hidden states and the original hidden states as the supervised signal. The loss can be formulated as below:

$$Loss_{distll} = \sum_{q=1}^{n-i-1} KL - divergence(\widetilde{P}_{n+q}^l, P_{q+i}^l)$$

3.4 Inference stage

In the inference stage, the process is to first generate some sequence candidates and then verify them, the purpose of the verification stage is to ensure the tokens consistency with the auto-regressive decoding, We construct multiple sequence candidates into a tree structure by merging their common ancestors, then we flat the tree into a sequence and construct attention mask correctly to keep their orders, finally we send the whole sequence into the LLMs to verify the candidates and generate the new candidates at the same time. Figure2 shows the inference stage of both Hidden transfer and Medusa.

3.4.1 Tree attention

When predicting the draft tokens for the following several steps, it becomes clear that the draft tokens belonging to different steps form a tree structure according to their order. The tree attention mechanism transforms this hierarchical tree into a linear sequence while preserving the original positional indices of each token. Moreover, it employs a specialized attention mask to ensure that a token only attends to its ancestors within the tree structure, thereby upholding the causal language model's properties. During inference, a pre-defined tree structure and parser facilitate the rapid transformation of token candidates between the tree and sequence representations, obviating the need for additional computations.

4 Experiment

4.1 Setup

We evaluate the our method on two different series of models with different size, including LLaMA-2-CHAT-13B (Touvron et al., 2023b), LLaMA-2-CHAT-7B (Touvron et al., 2023b) and Vicuna-13B (Chiang et al., 2023), Vicuna-7B (Chiang et al., 2023). We use greedy sampling strategy for the LLMs and the tokens generated using our method are identical to those generated by standard auto-regressive decoding theoretically. We divide the experiments into main experiments subsection, analytical and ablation study. The main experiments are conducted on all models and the analytical and ablation study only conducted on 7B model for simplicity.

In the main experiments subsection, we compare our end-to-end time acceleration ratio with Medusa (Cai et al., 2023) and Self-speculative decoding (Zhang et al., 2023) to show the effectiveness of our method(more details in Appendix), Because the self-speculative decoding method (Zhang et al., 2023) need to carefully select the transformer layers skipped for each model, and it doesn't offer the initial skip layers of 7B model in its open source code for their optimizer algorithm, so we only compare with them on LLaMA-2-Chat-13B and vicuna-13B, we use the acceleration ratio reported in their paper of LLaMA-2-Chat-13B for the Xsum dataset, and run its open source code to evaluate their effectiveness on the Gsm8K dataset and vicuna-13B. we use the same Tree structure and predict the next three draft tokens (exclude the next token generated by the original lm-head) for Medusa and our method, and we do hidden transfer for our method on the 25 30 35 layers respectively.

Analytical experiments subsection and ablation

Model	Decoding Algorithm	XSum	Gsm8k
LLaMA-2-Chat-13B	Auto-regressive	1.000×	1.000×
	Self-speculative (Zhang et al., 2023)	1.241×	1.216×
	Medusa (Cai et al., 2023)	1.325×	1.976×
	Ours	**1.532**×	**2.275**×
Vicuna-13B	Auto-regressive	1.000×	1.000×
	Self-speculative (Zhang et al., 2023)	1.125×	1.118×
	Medusa (Cai et al., 2023)	1.247×	1.869×
	Ours	**1.419**×	**2.150**×
LlaMA-2-Chat-7B	Auto-regressive	1.000×	1.000×
	Medusa (Cai et al., 2023)	1.465×	1.762×
	Ours	**1.816**×	**2.135**×
Vicuna-7B	Auto-regressive	1.000×	1.000×
	Medusa (Cai et al., 2023)	1.388×	1.732×
	Ours	**1.786**×	**2.219**×

Table 1: The acceleration ratio for both forward times and end-to-end time

study aim to verify our motivation, So we first compare the draft tokens prediction accuracy between our method and two baselines (e.g. Medusa heads and Early existing on different intermediate layers), result shows that we have the best prediction accuracy for the future draft tokens in a single forward propagation; To verify our motivation, we also analyze how the hidden states similarity between the pseudo hidden states predicted and the original hidden states changing along with the forward propagation and prove the **refinement** of transformer layers, finally we train multiple transfer projections on different layers for different transfer steps to explore how to select the transfer layers. Finally we also compare the transfer prediction accuracy of the second transfer step between the setting of first pseudo hidden states masked or not in the ablation study

All experiments are conducted on a single NVIDIA A100-80GB GPU and all implementations are based on PyTorch using HuggingFace's architecture (Wolf et al., 2020; Lhoest et al., 2021)

4.2 Datasets

We use ShareGPT dataset as our training dataset for all models, and we use the test split of Extreme Sum marization (XSum) (Narayan et al., 2018), Gsm8k (Cobbe et al., 2021) as our test dataset. ShareGPT is a multi-round conversations dataset comprises nearly 70,000 samples, We train one epoch for all the models. XSum (Narayan et al., 2018) is a dataset for evaluation of abstract single-document summary systems, it's test split has 11,334 samples. we only sample 1000 sentences followed (Zhang et al., 2023). The Gsm8k dataset (Cobbe et al., 2021) encompasses a collection of 8,500 linguistically varied, high-quality math word problems for grade school students, all of which were meticulously crafted by human authors, we use its whole test split with 1000 samples. All the two datasets are evaluated under 1-shot setting followed (Zhang et al., 2023).

4.3 Main Results

Table 1 shows that the acceleration ratio of our method is significantly better than other baselines in end-to-end time for all the test dataset(more details in appendix), it has at most **1.28x** acceleration ratio compared with Medusa, which is similar to our method. Using hidden transfer to predict the pseudo hidden states in the intermediate layer gain more benefit in the overall performance than using Medusa head to predict the token distribution directly, this improvement is more obvious in 7B models, and we find that the acceleration ratio on Gsm8k is higher because the answer in Gsm8k is more logical and predictable with more mathematical symbols.

4.4 Analytical Study

In this subsection, we conduct some analytical experiments to further verify the effectiveness and motivation of our method, the key idea of our method is to predict draft tokens more correctly in a single forward propagation by predicting the pseudo hidden states in intermediate layers. So we

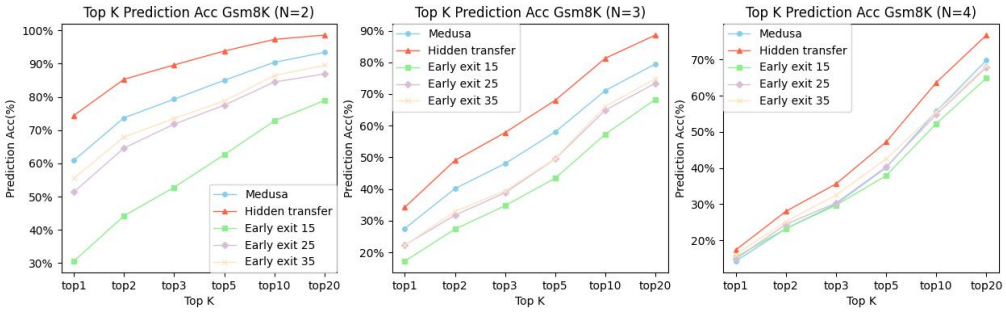

Figure 3: TopK tokens' prediction accuracy using three prediction methods on LLaMA-2-Chat-13B model including directly train different lm-heads on some intermediate layers (denoted as Early exit in the figure), Medusa method and our hidden transfer method (We transfer the pseudo intermediate hidden states of the next 3 tokens on the 25_{th}, 30_{th} and 35_{th} layers respectively), The N in the figure is the prediction step (N=2 means we predict the first draft token). It's clear that our method achieve the best prediction accuracy

first compare our method with Medusa and early exiting to show that we have better draft tokens prediction accuracy; then we analyze how the hidden states change during the forward propagation to verify the refinement we proposed. We also verify the draft tokens prediction dependency and show how to select the transfer layers.

Draft tokens prediction accuracy In a single forward propagation (given the token sequence X_1, ... X_n and model need to predict the future K draft tokens $\widetilde{X}_{n+2}$, ... $\widetilde{X}_{n+k+1}$), so we first compare the draft tokens' prediction accuracy on three different methods: early exiting in the intermediate layers, using medusa heads and our hidden transfer method. Early exiting method trains independent lm-heads in several intermediate transformer layers to directly predict the draft tokens(for example train a lm-head and use it to map h_n^j to $\widetilde{X}_{n+2}$), We use the LLaMA-2-Chat-13B as the base model for this experiment, three different methods are trained on ShareGPT dataset for one epoch and we set K as 3. We random sample 100 sequences from the test split of XSum and Gsm8K dataset respectively, for each sequence, we random split 50 points at its output part, and for each split-point, we use the token sequence before as the prompt input and predict K draft tokens using different methods and compare with the tokens generated greedily by the original model in the following K steps.(We choose five random seeds and average the results to better eliminate random errors) Figure 3 shows that our hidden transfer method achieve the best prediction accuracy among them.

Pseudo hidden states refinement we conduct another experiment to prove that the forward propagation in transformer layers can **refine** the predicted hidden states by given more semantic information using self-attention mechanism. We compare the cosine similarity of the pseudo hidden states predicted with the **original** hidden states, and trace how the cosine similarity changes with the forward process (for example, we denote the prompt sequence as X_1, ... X_n and we calculate the cosine similarity between pseudo hidden states $\widetilde{h}_{n+1}^t$ and real h_{n+1}^t, h_{n+1}^t is the hidden state of X_{n+1} on the t_{th} layer and X_{n+1} is the greedy decoding output token given the n prefix context). We random sample 100 sequences for two datasets and random split 50 times for each sequence as well. Figure 4 shows the cosine similarity get closer along with the forward process, which proves that with the forward process, the hidden states can be refined by the transformer layers.

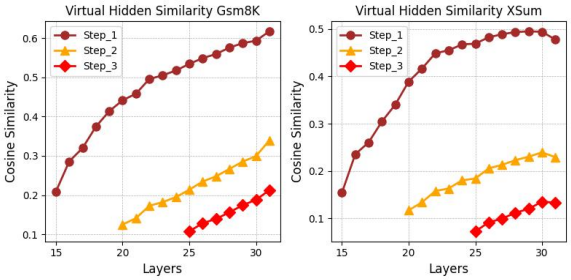

Figure 4: Hidden states similarity between the virtual hidden states predicted and the original hidden states.

How to choose transfer layers In the inference stage of our method, we need to choose which layer to transfer and generate the pseudo hidden states, there's a trade-off between accuracy and efficient: if we transfer on the lower layers, the pseudo hid-

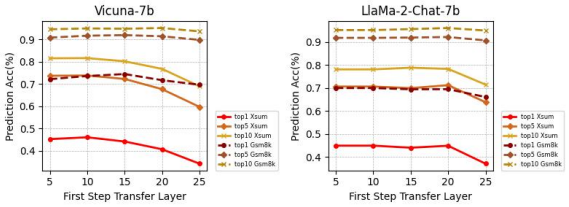

Figure 5: The first transfer step prediction accuracy on different layers for Vicuna-7b and LlaMa-2-Chat-7b. TopK means the topk tokens predicted by the transfer step include

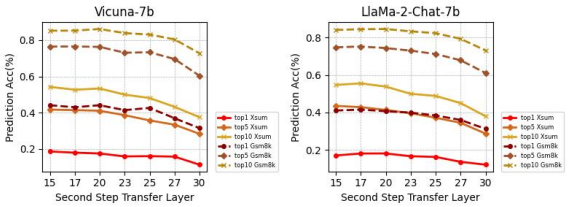

Figure 6: The second transfer step prediction accuracy on different layers for Vicuna-7b and LlaMa-2-Chat-7b with the fixed transfer step 15. TopK means the topk tokens predicted by the transfer step include

den states will pass more subsequent layers and take more computing resource but gain more semantic information by interacting with the hidden states of context; if we transfer on higher layers, the pseudo hidden states will take less computing resource with less semantic information. So we train the first and second transfer step on different layers of a fixed LLM to study the impact of the transfer layer selection. We choose Vicuna-7B and LlaMa-2-7b-chat model. For the first transfer step, we train different transfer structure on the 5_{th}, 10_{th}, 15_{th}, 20_{th}, 25_{th} layers seperately, and report the token prediction accuracy in figure 5, we found that from the lower layer to the middle layer, the prediction accuracy is basically unchanged, and from the middle layer to the high layer, the prediction accuracy drops rapidly, which proves that the middle layer to do tranfer is an optimal choice for both accuracy and computational efficiency, so we choose the 15_{th} layer to conduct the first transfer step. After fixed the first transfer layer, we also train different transfer structures on the layers higher than it as our second transfer layer. Figure 6 shows that the second transfer step have the same rule, starting from the 15_{th} layer, the accuracy rate remains unchanged within a range, and then rapidly decline, so we choose 20_{th} layer to conduct the second transfer step.

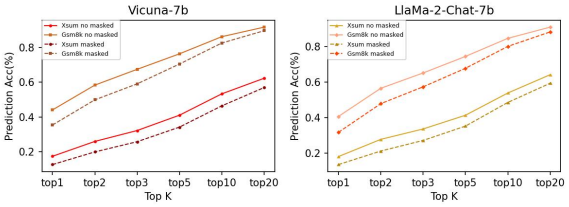

Figure 7: The second transfer step prediction accuracy under masked and no masked setting, we transfer the hidden states on 15_{th} and 20_{th} respectively

4.5 Ablation study

It's clear that Medusa (Cai et al., 2023) predict the draft tokens in parallel which means the generation between different draft tokens is independent. Our motivation is that serialized generation of draft tokens will gain better performance and we use experiment to prove it. We compare the prediction accuracy of the second transfer step under two setting: **Masked** and **No masked**, **Masked** means the second pseudo hidden state can not see the first pseudo hidden state in the forward process, and **No masked** is the normal self-attention mechanism. Figure 4 shows that in all the models and datasets, **Masked** performs worse which prove that the semantic information of the first draft token is important to the generation of the second draft token.

5 Conclusion

In this paper, we introduce a novel parallel decoding approach, named hidden transfer, designed for accelerating inference in large language models. By training a linear transformation projection in the intermediate layers, our model is capable of predicting the pseudo hidden states of multiple subsequent tokens in a single forward propagation. These predicted hidden states obtain additional semantic information through subsequent transformer layers, resulting in enhanced prediction accuracy. Through analytical experiments, we have proved that the hidden states predicted by the intermediary layers are progressively refined, gaining increased semantic information in the subsequent transformer layers by interacting with context. Our experiments demonstrate that our method outperforms existing approaches in terms of predictive precision within a single forward iteration and also achieves substantial gains in generation velocity.

Limitation

In the verification stage of our method, we simply utilized vanilla tree attention without specific optimization. However, the structural choices of tree attention significantly impact the generation speed. Therefore, future work will focus on optimizing it. Furthermore, we aim to conduct a more extensive set of analytical experiments to elucidate the underlying mechanisms of hidden transfer better and design improved training methodologies to enhance the quality of draft token generation. Our approach necessitates the expansion of the input sequence during both training and inference, which may lead to an increase in computational resource requirements. This issue will be addressed in the future research.

References

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. *arXiv preprint arXiv:2310.05424*.

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

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

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

Tianyi Chen, Tianyu Ding, Badal Yadav, Ilya Zharkov, and Luming Liang. 2023b. Lorashear: Efficient large language model structured pruning and knowledge recovery. *arXiv preprint arXiv:2310.18356*.

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

Jang Hyun Cho and Bharath Hariharan. 2019. On the efficacy of knowledge distillation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4794–4802.

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

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. 2019. Depth-adaptive transformer. *arXiv preprint arXiv:1910.10073*.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pages 10323–10337. PMLR.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

Song Guo, Jiahang Xu, Li Lyna Zhang, and Mao Yang. 2023. Compresso: Structured pruning with collaborative prompting learns compact large language models. *arXiv preprint arXiv:2310.05015*.

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

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

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. 2021. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *The Journal of Machine Learning Research*, 22(1):10882–11005.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. *arXiv preprint arXiv:2305.02301*.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713.

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.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, et al. 2021. Datasets: A community library for natural language processing. *arXiv preprint arXiv:2109.02846*.

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. *arXiv 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.

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

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

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

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

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Noam Shazeer. 2019. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*.

Benjamin Spector and Chris Re. 2023. Accelerating llm inference with staged speculative decoding. *arXiv preprint arXiv:2308.04623*.

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

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models.

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.

Hanrui Wang, Zhekai Zhang, and Song Han. 2021. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 97–110. IEEE.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2023. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*, 35:27168–27183.

Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. 2022. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*.

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*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Chris De Sa, and Zhiru Zhang. 2019. Improving neural network quantization without retraining using outlier channel splitting. In *International conference on machine learning*, pages 7543–7552. PMLR.

A Appendix

Decoding Algorithm	XSum 1000 sentences						
	tokens	forward	tokens/forward	forward acc rate	time	time/tokens	time acc rate
Auto-regressive	456830	456830	1.000	1.000	15016	0.0328s/token	1.000
Medusa	456107	259730	1.756	1.756	12039	0.0263s/token	1.247
Hidden transfer	456177	223535	2.040	2.040	10547	0.0231s/token	1.419
	Gsm8k 1000 sentences						
	tokens	forward	tokens/forward	forward acc rate	time	time/tokens	time acc rate
Auto-regressive	233040	233040	1.000	1.000	7687	0.0329s/token	1.000
Medusa	231708	111028	2.086	2.086	4096	0.0176s/token	1.869
Hidden transfer	231763	95537	2.425	2.425	3551	0.0153s/token	2.150

Table 2: The concise acceleration ratio including the time and forward times for Auto-regressive decoding, medusa decoding and hidden transfer decoding for vicuna-13b on Xsum and Gsm8k

Decoding Algorithm	XSum 1000 sentences						
	tokens	forward	tokens/forward	forward acc rate	time	time/tokens	time acc rate
Auto-regressive	420006	420006	1.000	1.000	14047	0.0334s/token	1.000
Medusa	420323	224678	1.870	1.870	10614	0.0252s/token	1.325
Hidden transfer	419289	190557	2.200	2.200	9171	0.0218s/token	1.532
	Gsm8k 1000 sentences						
	tokens	forward	tokens/forward	forward acc rate	time	time/tokens	time acc rate
Auto-regressive	153820	153820	1.000	1.000	5080	0.0330s/token	1.000
Medusa	152968	68649	2.228	2.228	2567	0.0167s/token	1.976
Hidden transfer	153031	59190	2.585	2.585	2228	0.0145s/token	2.275

Table 3: The concise acceleration ratio including the time and forward times for Auto-regressive decoding, medusa decoding and hidden transfer decoding for llama2-chat-13b on Xsum and Gsm8k

Decoding Algorithm	XSum 1000 sentences						
	tokens	forward	tokens/forward	forward acc rate	time	time/tokens	time acc rate
Auto-regressive	134806	134806	1.000	1.000	3621	0.0268s/token	1.000
Medusa	133757	79803	1.676	1.676	2585	0.0193s/token	1.388
Hidden transfer	134798	59753	2.255	2.255	2033	0.0150s/token	1.786
	Gsm8k 1000 sentences						
	tokens	forward	tokens/forward	forward acc rate	time	time/tokens	time acc rate
Auto-regressive	152012	152012	1.000	1.000	3853	0.0253s/token	1.000
Medusa	152903	76511	2.022	2.022	2235	0.0146s/token	1.732
Hidden transfer	153651	59616	2.577	2.577	1764	0.0114s/token	2.219

Table 4: The concise acceleration ratio including the time and forward times for Auto-regressive decoding, medusa decoding and hidden transfer decoding for vicuna-7b on Xsum and Gsm8k

Decoding Algorithm	**XSum 1000 sentences**						
	tokens	forward	tokens/forward	forward acc rate	time	time/tokens	time acc rate
Auto-regressive	327433	327433	1.000	1.000	8471	0.0258s/token	1.000
Medusa	327488	186063	1.760	1.760	5785	0.0176s/token	1.465
Hidden transfer	328083	145149	2.260	2.260	4683	0.0142s/token	1.816
	Gsm8k 1000 sentences						
	tokens	forward	tokens/forward	forward acc rate	time	time/tokens	time acc rate
Auto-regressive	176339	176339	1.000	1.000	4460	0.0252s/token	1.000
Medusa	174551	85251	2.047	2.047	2507	0.0143s/token	1.762
Hidden transfer	175219	70813	2.473	2.473	2073	0.0118s/token	2.135

Table 5: The concise acceleration ratio including the time and forward times for Auto-regressive decoding, medusa decoding and hidden transfer decoding for llama-2-chat-7b on Xsum and Gsm8k