Frontiers in Computing and Intelligent Systems
ISSN: 2832-6024 | Vol. 9, No. 3, 2024

Dynamic Layer Skipping for Large Language Models on Natural Language Understanding Tasks and Machine Translation Using Reinforcement Learning

Wei Xu [1, *], Xiaodong Jin [2]

[1] School of Humanities, College of Science and Technology, Ningbo University, Ningbo, Zhejiang, China
[2] School of Foreign Languages, Ningbo University of Technology, Ningbo, Zhejiang, China

*** Corresponding author:** Wei Xu (Email: xuwei2@nbu.edu.cn)

Abstract: Large Language Models (LLMs) demonstrate remarkable proficiency in various natural language processing (NLP) tasks. However, their extensive size, resulting from the inclusion of billions of parameters across multiple layers, presents significant challenges regarding storage, training, and inference. Traditional methodologies such as model pruning and distillation are employed to decrease the size of these models, but these techniques often result in a compromise on performance retention. In this work, we propose a novel framework that uses dynamic layer skipping for different samples to accelerate the inference speed of LLMs. First, we add an adapter layer at each transformer layer to predict whether to skip the next layer or not, and we propose layer skip pretraining to recover the model's performance. Second, we propose using reinforcement learning (RL) to optimize the model and design several strategies to stabilize the training. Extensive experiments on four natural language understanding (NLU) datasets and three machine translation datasets and ablation studies show that our method achieves SOTA performance among layer skipping methods on LLMs.

Keywords: Large Language Model; Reinforcement Learning; SOTA Performance.

1. Introduction

Since ChatGPT and GPT 4 [1], Large Language Models (LLMs) have become ubiquitous in almost all sub-fields of natural language processing. These models are pretrained on an enormous number of tokens [2] and have shown excellent capabilities in following instructions for sentence generation. For models with more than 10 billion parameters, emergent capabilities have occurred [3], such as in-context learning and reasoning capabilities [4]. An appealing property of LLMs is that after being trained with large amounts of instruction-response data, the models can generalize well across many different natural language processing tasks [5], such as natural language inference [6], information extraction [7][8][9], and text classification [10][11][12].

Despite the impressive capabilities of LLMs in completing instruction-following tasks [13][14][9], the computational cost of LLMs is a significant issue because training large models requires substantial time, computational resources, and considerable expenses. Furthermore, the large latency is an obstacle for LLMs to be applied in many industrial scenarios with low latency requirements, which require fast inference speed to meet users' needs.

There is much research work focusing on speeding up the inference of BERT [15] and LLMs, such as network pruning [16][17], student network distillation [18][19], quantization [20][21], and early exiting [22][23][24][25][26]. Network pruning, student network distillation, and quantization have been very effective in reducing the parameters or computation cost of pretrained language models, but the model architecture of these methods is usually fixed during inference. Some early exiting works [23][26] have proved that the network structure varies for different test samples during inference. They use dynamic inference methods, based on certain criteria such as the entropy of logits, to allocate different BERT layers for different test samples in inference stages. Therefore, due to its potential in applications, we focus on applying early exiting to LLMs.

Early exiting requires a multi-exit pretrained language model, such as BERT and GPT [27], with an intermediate classifier (or early exit layer) installed at each transformer layer. Then, a dynamic early exiting mechanism is applied at each layer during the forward pass to assess whether to exit at the current layer. Early exiting can work together with static model compression methods, such as performing early exiting on a pruned model [28]. Most of the work [22][23][25][26] on early exiting focuses on the classification task using BERT as the backbone model, and recently some work has started to apply early exiting to generative tasks.

Men et al. [29] found that LLM layers have high redundancy, and removing the redundant layers does not significantly drop LLM performance. CALM [30] adds an early exit layer at each decoder layer of a small T5 model (8 layers) [31] and uses entropy as the criterion to exit. However, for LLMs such as Llama [32] with a large language modeling head size and deep layers, it would incur significant computational costs. Skipdecode [33] proposes using layer skipping, employing rule-based methods, to gradually skip more intermediate GPT layers as more tokens are generated. This approach avoids installing an exit layer at each GPT layer. However, it applies the same skipping layers for all samples, which does not constitute dynamic inference. Yuan et al. [34] directly prunes the layers of LLMs, retains the last layer, and fine- tunes the model on NLU datasets. It shows that skipping more than half of the LLM's layers can still achieve comparable or slightly better results. However, these pruning methods do not consider dynamic inference for different samples. Some hard samples may need more layers to compute, while easy samples may need fewer layers. Also, certain layers may be important for some samples while

unimportant for others.

Since previous work has not applied dynamic inference to LLMs, we propose a novel framework for speeding up LLMs by using controllers to dy namically skip layers for different samples. We use reinforcement learning (RL) to train the controller, which decides whether to dynamically skip the next layer. Since the training of RL is unstableand hard to converge, we design several strategies such as introducing adapters and layer skipping pre-training. We also refine the sampling strategy of RL by not skipping layers that are dissimilar to previous layers. We conduct experiments on natural language understanding (NLU) datasets and machine translation (MT) task, formulating this task as a prompt-based language modeling task using instructions. The experiments on four benchmark NLU datasets and three MT datasets show that our work achieves state-of-the-art performance compared to previous work.

Our contributions are as follows:

(1) To the best of our knowledge, we are the first to apply a dynamic inference method to large language models to achieve LLM inference speed-up.

(2) We propose a novel framework that uses an adapter layer to dynamically skip LLM layers without introducing high computational costs.

(3) We propose using reinforcement learning to optimize our framework, and we introduce some sampling strategies to improve the training of RL.

(4) We conduct experiments to show that our method achieves SOTA performance for LLM inference speed-up. In addition, ablations and intrinsic evaluations demonstrate the effectiveness of our proposed method.

2. Preliminaries

Due to the length limit, readers are referred to Appendix A.1 for LLM applications on natural language understanding (NLU) datasets.

2.1. Layer Skipping

Skip-decode [33] is the first to propose a layer skipping method, which can be viewed as another form of early exiting. The main difference is that layer skipping can bypass any layer, while early exiting exits at a specific layer and stops computing the subsequent layers. However, it is very challenging to apply early exiting to LLMs, as early exiting methods install an early exit layer, which is a classifier layer, at each Transformer layer to compute the entropy of the logits and predict whether to exit. However, for LL Ms, the classifier layer would be a language modeling (LM) head classifier, typically larger than 100,000 units. Additionally, since LLMs often have many deep layers, such as 36 or 48, installing an LM head at each layer would incur unbearably high computational costs. Therefore, Skip-decode directly assigns the number of layers to skip instead of assessing each Transformer layer. The model gradually skips more intermediate layers as it generates more tokens. It applies the same skipping layers for all samples, which is not dynamic. Additionally, this approach assumes that the tokens at the end of the responses are simpler than those at the beginning. A counterexample is using LLMs for the named entity recognition task, where the LLM generates many entities mentions as results. However, the entity mentions generated at later steps are just as important as those at earlier steps, making it unfair to assign fewer layers to these tokens at later generation steps. Also, Skip-decode skips the shallow layers first and then the deep layers. Much work [29][24][35] has demonstrated that the later layers of BERT or LLMs have more redundancy compared to earlier layers. Thus, we aim to develop an adaptive method to automatically determine which layers to skip for predicting each token.

2.2. Problem Statement

We apply prompts to use LLMs on NLU and MT datasets. For each dataset, we design a specific prompt, and during training, we combine all datasets to train one model. We show our prompt design in Appendix A.2. The prompt is the input to LLMs that contains the instructions and the input sentence to be classified. Denote the prompt as $[w_0, w_1, w_2, \dots, w_{n-1}]$, and the LLM we use has L layers. For generating responses, the prompt sequence will be embedded in the embedding layer:

$$H^0 = [h_0^0, h_1^0, \dots, h_{n-1}^0] = \text{Embedding}([w_0, w_1, \dots, w_{n-1}]) \quad (1)$$

and go through the transformer layers:

$$H^i = \left[h_0^{i+1}, h_1^{i+1}, \dots, h_{n-1}^{i+1}\right] = \text{Trans}^{i+1}\left(\left[h_0^i, h_1^i, \dots, h_{n-1}^i\right]\right) \quad (2)$$

for $i = 0,1, \dots, L-1$. At the final layer, the hidden representation h_{n-1}^L will be fed into the language modeling head to predict the next token by calculating its probability distribution with the language modeling head:

$$p(w_n \mid w_0, w_1, w_2, \dots, w_{n-1}) = \text{LMH}(h_{n-1}^L) \quad (3)$$

We use the cross entropy loss to fine-tune the LLMs:

$$\mathcal{L}_{\text{cls}} = -\sum_{i=1}^{C} y_i \log p_i \quad (4)$$

Note that with the attention mechanism of Transformer-based models, h_{n-1}^L contains information about the tokens in the previous steps. $p(\ )$ denotes the calculated distribution over the vocabulary, y is the ground truth. With a given decoding algorithm like beam search or nucleus sampling [36], one can select the predicted next token w_n, and the input sequence will be augmented to $x = [w_0, w_1, w_2, \dots, w_{n-1}, w_n]$.

The above process will continue until the end-of-sentence token is met. Note that in practice, one will not let the new sequence $[w_0, w_1, w_2, \dots, w_{n-1}, w_n]$ go through all the Transformers repeatedly since this requires too much computation. For efficient token generation, one will adopt the key-value (KV) caching mechanism, which is an efficient implementation for token generation in Transformer models. With the causal mask, the newly generated token will not affect the hidden states of the previous shorter sequences. Thus, we only need to calculate the key and value of the new token. The model can significantly reduce redundant computations during subsequent steps by storing the computed keys and values for previously processed tokens. However, in scenarios of early exiting, for example, if the next token exits later than the previous one, we need to recompute the KV values for the previous tokens. Skip-decode [33] then proposes to monotonically decrease exit points to eliminate the necessity to recalculate KV caches for preceding tokens.

3. Methodology

3.1. Dynamic Layer Skipping Architecture and Pretraining

Skip-decode [33] directly skips the layers of LLMs without training, lacking investigation into training methods that enhance the performance of layer skipping. Yuan et al. [34]

directly cuts off some layers and then fine-tunes the model. Both of these methods are static layer skipping. In the following part, we will introduce the architecture we use for dynamic layer skipping and the method we used to pretrain the model. The pretraining is designed to enhance LLMs' ability for dynamic layer skipping and to initialize the parameters of the controllers.

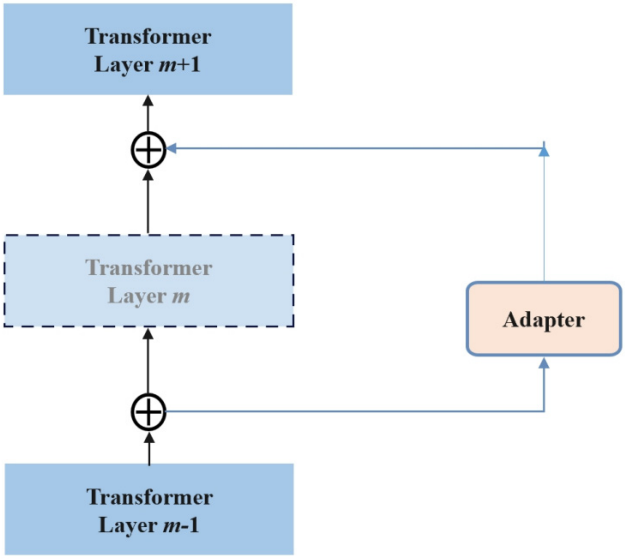


Figure 1. The adapter enhanced Transformer layers. We add an adapter layer at each transformer layer, if current transformer layer is skipped, then the hidden state goes through the adapter layer

Dynamic layer skipping architecture: We insert an adapter layer in parallel with each transformer layer, just like LoRA [37], as shown in Figure 1. This adapter consists of a two-layer MLP with a GELU activation and takes the hidden states h^i_{n-1} of w_{n-1} as the input feature and outputs the hidden states $h^{i\prime}_{n-1}$. Since the adapters use a bottleneck architecture [38], they are computationally efficient. For a transformer layer i, we input the hidden states of the last layer h^{i-1}_j into the transformer layer and the adapter layer, and we sum them to get the output hidden states of layer i:

$$h^i_j = \text{Adapter}^i\left(h^{i-1}_j\right) + \text{Trans}^i\left(h^{i-1}_j\right) \quad (5)$$

We use the adapter layer Adapter^{i-1} as the controller to decide whether to skip transformer layer i. By taking the output of the adapter layers as input for the next layer, the adapters learn the essential information needed to pass to the next layer. This information can greatly help the adapter assess whether to skip the next layer. By pretraining the adapters, it provides a good initialization of the adapter parameters compared to basic initialization methods such as Xavier initialization. A good initialization offers a smaller search space for reinforcement learning, and thus helps RL converge to the optimal policy.

Layer skipping pretraining: To train a model with layer skipping capabilities, we assign a random 0,1 mask m_i (i.e., a Bernoulli random variable) with a probability of being 0 set to $p(0 < p < 1)$ for each layer i. Thus, the above equation becomes:

$$h^i_j = \text{Adapter}^i\left(h^{i-1}_j\right) + m_i * \text{Trans}^i\left(h^{i-1}_j\right) \quad (6)$$

That is, if layer i is skipped, the hidden states will only go through Adapter^i. During training, the parameters of Transformer layer Trans^i can be either fully tuned or tuned with LoRA [37] to save computational resources.

Note that separate forward steps may result in different representations with random layer masks. We can add a consistency regularization objective to the CE loss term to ensure model consistency after random layer skipping. For the separate forward passes of the same input $[w_0, w_1, w_2, \ldots, w_{n-1}]$, one can obtain three different distributions P^0, P^1, and P^2 for w_n. P^0 denotes the teacher model that does not skip, while P^1 and P^2 are student models that randomly skip different layers. We use KL-Divergence to provide consistency regularization:

$$\mathcal{L}_{\text{reg}} = \text{KL}(P^1, P^2) \quad (7)$$

, and distill knowledge from teacher model:

$$\mathcal{L}_{\text{distill}} = \text{KL}(P^0, P^1) + \text{KL}(P^0, P^2) \quad (8)$$

The final loss for pretraining is:

$$\mathcal{L}_{\text{pretrain}} = \alpha * \left(\mathcal{L}_{\text{distill}} + \mathcal{L}_{\text{reg}}\right) + \mathcal{L}_{\text{cls}} \quad (9)$$

, where α is the hyper-parameter controlling the weight of loss.

3.2. Reinforcement Learning Framework

We now propose a reinforcement learning (RL) based approach for dynamically determining which layers to skip when predicting the next token w_n given $w_{n-1}, \ldots, w_0$. At each Transformer layer i, we use the adapter layer to get the hidden states $h^{i\prime}_{n-1}$, then we add an RL-specific head, which is a one-layer MLP with sigmoid activation, to the adapter. Thus, the controller consists of an adapter layer and an RL-specific head. We use the output of the controller as the predicted probability of skipping, p^{skip}_{n-1}. If p^{skip}_{n-1} exceeds a threshold, we will skip the next layer. Note that we do not have any ground truth labels for each layer's controller. Thus, the controllers are optimized via reinforcement learning algorithms like REINFORCE [39].

To be more specific, we define the state, action, reward, advantage function, and objective function below.

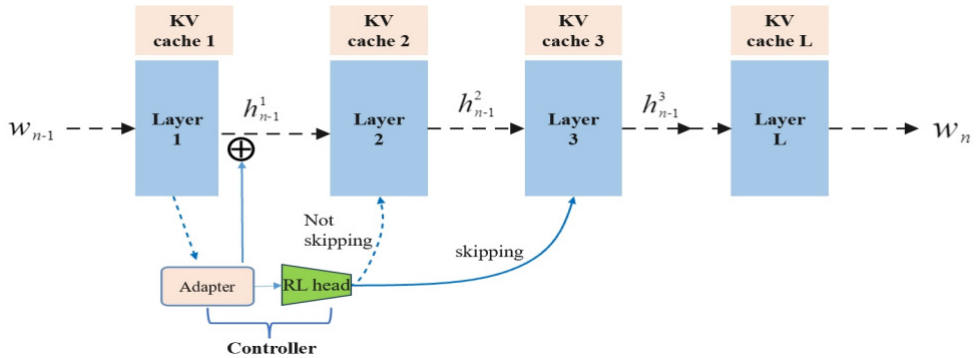


Figure 2. Dynamic layer skipping framework: At each transformer layer we have a controller, which consists of an adapter layer and a RL head. We use the output of the controller to determine whether to skip computing the next transformer layer

State State s_l consists of the sequence representations from transformer layer l.

Action The actions at each layer are Skip, Select. Skip means to skip computing the hidden states of the next layer, while Select means the hidden states go through the next transformer layer to compute.

The controllers we use, which are called policy networks in RL, generate probabilities for actions and contain an adapter layer and an RL-specific head. More formally, the policy network is defined as:

$$\pi_\theta(a_l \mid s_l) = \sigma\Big(W_3\Big(\mathrm{GELU}\big(W_2\big(\mathrm{GeLU}\,(W_1 H_{\text{last}} + b_1)\big) + b_2\big)\Big)\Big) \quad (10)$$

where σ is the sigmoid function, a_l denotes the action at state s_l, l is the current layer, H^l last denotes the last token representation at layer l, and θ are the parameters of the policy network. We will use $A = [a_1, a_2, \ldots, a_L]$ to denote the action trajectory the policy takes up to layer L.

Reward Denote the ground truth of token w_n as w_n^* (provided in the training data). At an intermediate layer l, if the layer is not skipped, then the reward is $r(a_l \mid s_l) = -1$. If an intermediate layer l is skipped, the reward is $r(a_l \mid s_l) = 1$. At the final layer, after obtaining the probability distribution from the LM head as $P(w_n \mid h_{n-1}^L)$, we will receive the final reward $-\mathrm{CE}\left(w_n^*, P(w_n \mid h_{n-1}^L)\right)$. The cumulative intermediate reward is $l_n = \sum_{l=1}^{L} r(a_l \mid s_l)$. Thus, the total reward for the controller is:

$$R(A \mid \theta) = -\mathrm{CE}\left(w_n^*, P(w_n \mid h_{n-1}^L)\right) - \lambda * l_n \quad (11)$$

Where l_n is the number of transformer layers actually processed, we encourage the model to skip as many layers as possible. λ is a hyperparameter for controlling the penalty of computational costs. $\mathrm{CE}(\ )$ is the cross-entropy loss, reflecting the performance of layer skipping. By maximizing the reward, the controllers learn to skip layers while maintaining accurate token predictions.

It should be noted that the computation of the reward is based only on the LM-logits of the last layer because if we were to compute the reward for each layer, we would need to assign an individual LM head at each layer, which would require a significant number of computational resources.

Sampling strategy Since we cannot directly optimize the reward of each layer, in the early training stages of RL, the search space would be very large, making it difficult for the model to converge to the optimal policy. To address this problem, we use the following strategies to stabilize the reinforcement learning:

(1) We do not drop the first layer and the last layer, because it would cause a large performance drop.

(2) We calculate the cosine similarity between two consecutive layers. Zhang et al. [26] found that in the BERT model, if the predictions of several consecutive layers have high similarity, then redundancy may exist in those layers, allowing for early exiting. Based on this, we skip those layers if their previous layer is similar to the earlier layers. In our setting, the similarity between layer i and layer $i-1$ is calculated by the cosine similarity of the last token hidden states: $\cos_i = \cos\left(h_{\text{last}}^i, h_{\text{last}}^{i-1}\right)$. If $\cos_i$ is higher than a threshold τ, we say layer i is $s-p$, meaning it's similar to its previous layer. For a layer i, if its M previous layers are all $s-p$, it indicates that the model's hidden states have not changed much for M consecutive layers. Then, for the skipping prediction probability p^{i+1} skip for the next layer $i+1$, we add a small value t to increase its probability of exiting at the next layer, such that p^{i+1} skip $= t + p_{\text{skip}}^{i+1}$. During training, we gradually reduce the number M to 0, where 0 means that we won't increase the probability for any layer. Initially, M is set at 3. Therefore, at the initial stage of RL training, we encourage the model to skip more confident layers by measuring their similarities with previous layers, thereby eliminating the sampling trajectories and stabilizing RL training.

Objective function The goal of reinforcement learning is to optimize the policy network to maximize the expected reward. Formally, the objective function is defined as follows:

$$J(\theta) = E_{(s_l, a_l)\sim\pi(a_l \mid s_l;\theta)} r[(s_1, a_1) \ldots (s_L, a_L)] \quad (12)$$

where L is the total number of layers, representing the number of states. According to the REINFORCE algorithm [39] and the policy gradient method [40], we update the network with the policy gradient as follows:

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} R \cdot \nabla_\theta \log\, \pi_\theta(a_t \mid s_t) \quad (13)$$

3.3. Model Training

Here we present our entire training procedure:

(1) Adapter Initialization: Fine-tune the adapter layer on instruction-tuning datasets while freezing the parameters of the LLM model. This step helps initialize the parameters of the adapter layers.

(2) Skip Pretraining: Train the parameters of the LLM and adapter layers on NLU and MT datasets using $\mathcal{L}_{\text{pretrain.}}$. One can choose to update the full parameters of the LLMs or use parameter-efficient tuning such as LoRA [37] and QLoRA [41]. This step enhances the model's ability to perform layer skipping and also provides a good initialization for reinforcement learning.

(3) Reinforcement Learning: Conduct RL on NLU and MT datasets. Freeze the parameters of the LLMs and LoRA, and update the parameters of the controller, which includes the adapter layer and RL-specific head. This step trains the controller's ability to perform layer skipping.

(4) Fine-tuning: Unfreeze the parameters of the LLMs or LoRA and the adapter, and train the entire model with the task-specific objective and the RL objective simultaneously. Specifically, we adopt the training method used in MIXER [42], gradually reducing the weight of the task-specific objective.

3.4. Inference with Adaptive Layer Skipping

During inference, at layer $l-1$, we use the controller installed at this layer to predict whether to skip the next layer l. We use the hidden states of layer $l-1$ as the hidden states for layer l. Additionally, as discussed previously, with KV caching, the next token depends on the KV caches of the current and previous tokens. Skip-decode [33] monotonically decreases exit points, but we believe this approach may restrict layer skipping decisions and result in performance degradation. Thus, we propose: when a layer i is skipped for predicting w_{n+k}, the KV caches of w_{n+k-1} at layer l are filled with those from layer $l-1$.

To improve the efficiency of inference, one can also perform layer skipping at the batch level. Specifically, a layer will be skipped if the predicted probabilities for most of the samples in the batch (such as more than half) exceed a

threshold of 0.5. Thus, for inference, we can strike a balance: process one sample at a time to assign different skip layers for different samples to achieve more accurate results, or process a batch at a time to share the same skip layer for a batch of samples to infer in parallel. Moreover, we can adjust the threshold to skip more or fewer layers to achieve different speed-up ratios.

4. Experiments

4.1. Datasets

We use four NLU datasets from the GLUE benchmark [43]: RTE (Recognizing Textual Entailment): Determines whether one sentence logically follows from another. MRPC (Microsoft Research Paraphrase Corpus): Identifies whether two sentences are semantically equivalent. CoL A (Corpus of Linguistic Acceptability): Assesses whether an English sentence is linguistically acceptable. SST-2 (Stanford Sentiment Treebank): Evaluates the sentiment of a sentence as positive or negative.

We use three machine translation datasets: WMT 2023 (English-Chinese): Focuses on translating between Chinese and English, particularly in news articles and other formal content. It is a key benchmark for Chinese-English translation quality.

WMT 2023 (English-German): Evaluates machine translation systems on translating news articles between English and German. It is a major benchmark in the translation community.

IWSLT 2023 (German-English): Focuses on spoken language translation, particularly for translating TED Talks and other speech data from German to English.

4.2. Baseline Methods

We compare our method with the following baselines:

Vanilla model: We directly fine-tune the LLM on NLU and MT datasets without skipping any layers.

Skip-decode [33]: Following its method, we skip the intermediate layers of LLMs. As more tokens are generated, the number of skipped layers increases. The method does not involve training the LLMs; we replicate its method and test it on NLU and MT datasets.

Static layer skipping [34]: This method directly skips the intermediate layers of LLMs while keeping the last layer. It then finetunes the LLMs on NLU and MT datasets.

Layer skip pretraining: We skip the same set of layers as the static layer skipping baseline method and fine-tune the LLMs on NLU and MT datasets using the layer skip pretraining method proposed in this work.

4.3. Experimental Settings

Devices: We implement our method based on Hugging Face's Transformers [44]. We conduct our experiments on two Nvidia V100 32GB GPUs.

PTM models We adopt the following LLMs for our experiments: GPT2-XL [45], OPT-1.3B [46], Falcon-7B [47], Llama2-7B-Chat [32]. Readers are referred to Appendix A.3 for the details of these LLMs.

Training settings for all training stages of our experiments, we adopt the following settings: We use QLoRA [41] to train the LLM backbone model, with the LoRA rank set to 16. The maximum sequence length is 1024, and we adopt the efficient sequence packing strategy [48] to accelerate training. The batch size is set to 64. We employ AdamW [49] as the optimizer. For different training stages, we adopt different learning rates and epochs: In the Adapter Initialization stage, the epoch is set to 1 and the learning rate to $1e-4$. In the Skip Pretraining stage, the epoch is set to 5 and the learning rate to 5e-5. In the Reinforcement Learning stage, the epoch is set to 10 and the learning rate to $2e-5$. In the Fine-tuning stage, the epoch is set to 5 and the learning rate to $1e-5$. For each method, we use different settings to instruct the model to skip more or fewer layers, and we obtain the results of these different settings. In detail, for Skip-decode, we set the maximum ratio of skipped layers to total layers at $90\%, 70\%, 50\%, 30\%$, and 10%; for the Yuan et al. [34] method, we follow the same settings as Skip-decode; for our method, we set the thresholds for skipping layers at 0.8,0.7,0.5,0.4, and 0.3.

4.4. Main Results

Table 1. F1-score (%) of methods using the Llama-7B-Chat backbone on NLU tasks across five random seeds. AVG represents the average score achieved by skipping different numbers of layers, and BEST represents the best score among all settings.

	RTE		MRPC		CoLA		SST-2	
	AVG	BEST	AVG	BEST	AVG	BEST	AVG	BEST
Vanilla model	77.1	77.1	89.8	89.8	57.7	57.7	91.9	91.9
Skip-decode	44.7	48.9	69.1	73.2	41.6	47.5	81.0	82.5
Yuan et al. [34]	62.8	68.6	76.7	82.9	48.7	51.3	86.2	89.1
Layer skip pretraining	64.7	73.1	79.0	85.2	51.2	56.3	88.7	91.3
Ours(bs=1)	68.8	**77.5**	85.6	**90.5**	49.4	**58.1**	90.6	**92.7**
Ours(bs=8)	67.7	77.1	85.0	90.3	46.7	57.2	90.3	92.5
Ours(bs=16)	66.5	76.5	84.7	89.7	45.8	56.5	90.0	92.1

To validate the efficacy of our proposed method, we conducted comparative experiments against baseline layer skipping or pruning methods. The experimental results are presented in Table 1 and Table 2. '$\mathrm{bs}=1$' denotes we do dynamic layer skipping for each sample, '$\mathrm{bs}=8$ or $\mathrm{bs}=16$' means we use batch-level skipping: all samples in a batch skip

the same layers. AVG represents the average score from different layer skipping settings mentioned in the training settings section, and BEST represents the best score among all settings. We set different thresholds to control the number of layers to be skipped.

We present the results of other LLMs in Appendix A.3.1. As we can see, by fine-tuning on a text-classification dataset, Yuan et al. [34] achieves better results with more skipped layers compared to Skip-decode. The layer skip pretraining method also boosts the performance of LLMs. Our method significantly outperforms the Yuan et al. [34] method, which applies the same layer settings for all samples, validating the effectiveness of dynamic layer skipping.

Table 2. BLEU score (%) of methods using the Llama-7B-Chat backbone on MT tasks across five random seeds. AVG represents the average score achieved by skipping different numbers of layers, and BEST represents the best score among all settings.

	WMT 2023		WMT 2023		IWSLT 2023	
Language Pair	English-chinese		English-German		German-English	
	AVG	BEST	AVG	BEST	AVG	BEST
Vanilla model	38.7	38.7	37.8	37.8	33.2	33.2
Skip-decode	32.8	34.6	36.4	37.1	30.8	33.1
Yuan et al. [34]	35.6	37.3	38.2	33.2	33.0	36.1
Layer skip pretraining	37.5	37.9	38.5	34.1	33.3	34.1
Ours(bs=1)	39.1	**39.8**	38.2	**38.9**	35.0	**35.9**
Ours(bs=8)	38.5	38.9	37.8	38.0	34.7	35.2
Ours(bs=16)	38.0	38.4	37.7	38.1	34.7	35.0

4.5. Discussions and Ablation Studies

We present the F1-score of the Llama2-7B-Chat model with different layer skipping ratios on the QNLI dataset in Figure 3. The layer skipping ratio is computed by dividing the number of skipped layers by the total number of layers. The results show that skipping nearly 30% of the layers can even improve performance, demonstrating the redundancy in LLM layers. The reasons why skipping many layers allows LLMs to still achieve comparable results are twofold: (1) much work [29][50][16] proves that the parameters of LLMs are redundant. (2) The parameters of LLMs can be further reduced if they are used only for certain tasks, such as NLU tasks. LLMs do not need the extra parameters to store knowledge and make complex inferences for solving other tasks such as code completion and multi-turn question answering.

4.5.1. Ablation on Dynamic Layer Skipping

For a LLM with L layers, we initialize an array of length L and fill it with 0; for the skipped layer, we set its index in the array to 1. Then, we can obtain arrays $a^1, a^2, \ldots, a^N$, where N is the total number of samples. Since the length of an array a^i is L, for each position l in a^i, we can compute the variance for all arrays, and we average the variance for all positions by:

$$\bar{\mu}_l = \frac{1}{L}\sum_{i=1}^{N} a_l^i \tag{14}$$

$$\sigma = \frac{1}{L}\sum_{l=1}^{L} \sqrt{\frac{1}{N}\sum_{i=1}^{N} \left(a_l^i - \bar{a}_l\right)^2} \tag{15}$$

where a_l^i the l position number in i-th array, $\bar{\mu}_l$ is average of all arrays at position l.

σ denotes the overall difference among all samples; a large variance indicates that the differences in the data are larger and suggests that the layer skipping options for different samples vary. We present the variance for different datasets in Table 3. As we can see, all have a large variance, indicating that the skipped layers for different samples vary significantly, which validates the importance of applying dynamic layer skipping. In contrast, the variance for static layer skipping is 0, because the same settings are applied to all samples.

Table 3. Variance of layer skipping for different datasets

Dataset	Variance
RTE	0.35
MRPC	0.21
CoLA	0.43
WMT2023	0.32
IWSLT2023	0.41

4.5.2. Ablation on Layer Skipping Pretraining

Layer skip pretraining enhances the model's ability to infer with fewer layers, providing a good initialization for the model to engage in reinforcement learning. As the results in Table 1 and Table 2, Yuan et al. [34] shows better performance than Skip-decode [33]. This indicates that further training can significantly recover the model's performance lost due to skipping layers. Furthermore, the results show that the model after layer skip pretraining achieves even better results than Yuan et al. [34], validating the effectiveness of the consistency loss and distillation loss from the teacher model.

4.5.3. Ablation on Sampling Strategy

Encouraging the model to skip more confident layers (measuring their similarities with previous layers) at initial steps can help the model find a better policy early on and converge to a better overall policy. As shown in Figure 4, we

can see that without the sampling strategy, the loss at initial stages is very unstable and may even increase significantly. This occurs because the model may skip some vital layers, and each skipping decision does not directly contribute to the final token prediction, leading the model to potentially learn a false policy. Thus, the sampling strategy helps make the training of RL more stable. Since the total number of sampling trajectories is 2^L, adding a sampling strategy can significantly reduce the number of low-quality trajectories, ensuring the model remains at a relatively good policy. We can also see that the final loss is lower, indicating the effectiveness of the proposed sampling strategy.

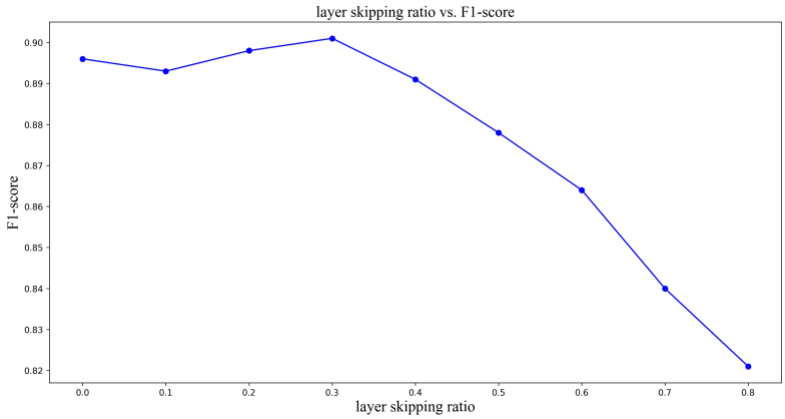
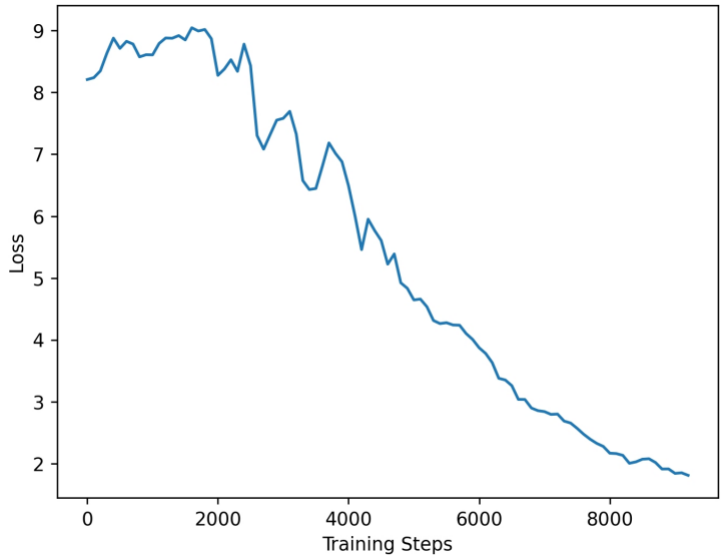
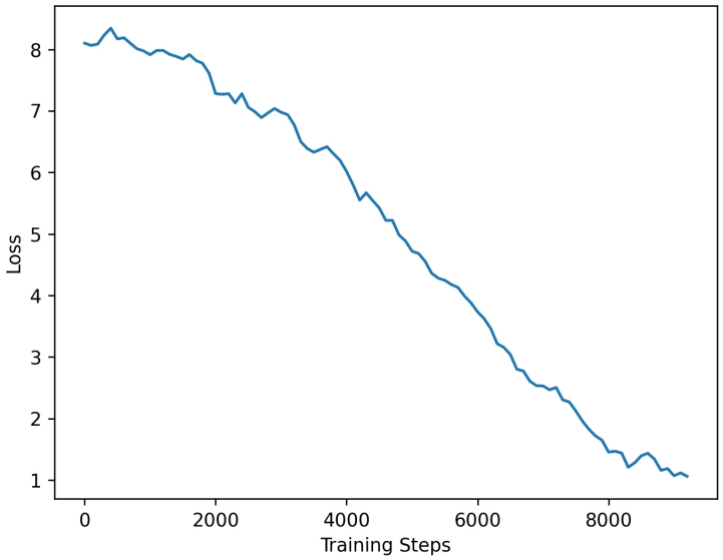


Figure 3. F1-score of Llama-7B-Chat model on QNLI dataset with different layer skipping ratios.



a) Loss without sampling strategy



b) Loss with sampling strategy

Figure 4. The variation diagram of RL loss with the increase of training steps

5. Conclusion

In this work, we present a novel approach for speeding up large language model (LLM) inference through the application of dynamic inference methods. To the best of our knowledge, we are the first to explore the dynamic layer skipping technique in the context of LLMs. Our contributions include the introduction of a framework that utilizes an adapter layer to dynamically skip LLM layers without introducing significant computational costs. Furthermore, we optimize our framework using reinforcement learning, incorporating sampling strategies to enhance the training process. Experimental results demonstrate that our method achieves state-of-the-art performance in terms of LLM inference speedup. Additional ablation studies and intrinsic evaluations further validate the effectiveness of our proposed technique.

References

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

[2] 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.

[3] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022. Emergent abilities of large language models. arXiv preprint arXiv:2206.07682.

[4] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits rea soning in large language models. Advances in neural information processing systems, 35:24824-24837.

[5] Hyung Won Chung, Le Hou, Shayne Longpre, BarretZoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. arXiv preprint arXiv:2210.11416.

[6] Xuanli He, Yuxiang Wu, Oana-Maria Camburu, Pasquale Minervini, and Pontus Stenetorp. 2023. Using natural language explanations to improve robust ness of in-context learning for natural language infer ence. arXiv preprint arXiv:2311.07556.

[7] Ankur Goswami, Akshata Bhat, Hadar Ohana, and Theodoros Rekatsinas. 2020. Unsupervised relation extraction from language models using constrained cloze completion. arXiv preprint arXiv:2010.06804.

[8] Xin Xu, Yuqi Zhu, Xiaohan Wang, and Ningyu Zhang. 2023. How to unleash the power of large lan guage models for few-shot relation extraction? arXiv preprint arXiv:2305.01555.

[9] Derong Xu, Wei Chen, Wenjun Peng, Chao Zhang, Tong Xu, Xiangyu Zhao, Xian Wu, Yefeng Zheng, and Enhong Chen. 2023. Large language models for generative information extraction: A survey. arXiv preprint arXiv:2312.17617.

[10] Xiaofei Sun, Xiaoya Li, Jiwei Li, Fei Wu, Shang- wei Guo, Tianwei Zhang, and Guoyin Wang. 2023. Text classification via large language models. arXiv preprint arXiv:2305.08377.

[11] Aristides Milios, Siva Reddy, and Dzmitry Bahdanau. 2023. In-context learning for text classification with many labels. In Proceedings of the 1st GenBench Workshop on (Benchmarking) Generalisation in NLP, pages 173-184.

[12] Gaurav Sahu, Olga Vechtomova, Dzmitry Bahdanau, and Issam H Laradji. 2023. Promptmix: A class boundary augmentation method for large language model distillation. arXiv preprint arXiv:2310.14192.

[13] Lochan Basyal and Mihir Sanghvi. 2023. Text summarization using large language models: Acomparative study of mpt-7b-instruct, falcon-7binstruct, and openai chat-gpt models. arXiv preprintarXiv:2310.10449.

[14] Youngjin Chae and Thomas Davidson. 2023. Large lan guage models for text classification: From zero-shot learning to fine-tuning. Open Science Foundation.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understand- ing. arXiv preprint ar Xiv: 1810.04805.

[16] Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and DanqiChen. 2023. Sheared llama: Accelerating language model pre-training via structured pruning. arXiv preprint arXiv:2310.06694.

[17] Ziqing Yang, Yiming Cui, Xin Yao, and Shijin Wang. 2022. Gradient-based intra-attention pruning on pre-trained language models. arXiv preprint arXiv:2212.07634.

[18] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. arXivpreprint arXiv: 1910. 01108.

[19] Chenhe Dong, Guangrun Wang, Hang Xu, Jiefeng Peng, Xiaozhe Ren, and Xiaodan Liang. 2021. Efficient bert: Progressively searching multilayer perceptron via warm-up knowledge distillation. arXiv preprint arXiv:2109.07222.

[20] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. 2023. Awq: Activation aware weight quantization for 11 lm compression and acceleration. arXiv preprint arXiv:2306.00978.

[21] 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.

[22] Surat Teerapittayanon, Bradley McDanel, and Hsiang Tsung Kung. 2016. Branchynet: Fast inference via early exiting from deep neural networks. In 2016 23 rd international conference on pattern recognition, pages 2464-2469. IEEE.

[23] Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Haotang Deng, and Qi Ju. 2020. Fastbert: a self- distilling bert with adaptive inference time. arXivpreprint arXiv:2004.02178.

[24] Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. 2020. BERT loses patience: Fast and robust inference with early exit. Advances in Neural Information Processing Systems. arXiv:2006.04152.

[25] Zhen Zhang, Wei Zhu, Jinfan Zhang, Peng Wang, Rize Jin, and Tae-Sun Chung. 2022. Pcee-bert: accelerating bert inference via patient and confident early exiting. In Findings of the Association for Computational Linguistics: NAACL 2022, pages 327-338.

[26] Jingfan Zhang, Ming Tan, Pengyu Dai, and Wei Zhu. 2023. Leco: Improving early exiting via learned exits and comparison-based exiting mechanism. In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop), pages 298-309.

[27] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language under-standing by generative pre-training.

[28] Bowen Shen, Zheng Lin, Yuanxin Liu, Zhengxiao Liu,Lei Wang, and Weiping Wang. 2022. Cost-eff: Col laborative

optimization of spatial and temporal effi ciency with slenderized multi-exit language models. arXiv preprint arXiv: 2210.15523.

[29] Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. 2024. Shortgpt: Layers in large language models are more redundant than you expect. arXiv preprint arXiv:2403.03853.

[30] Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler.2022. Confident adaptive language modeling. Ad vances in Neural Information Processing Systems. arXiv:2207.07061.

[31] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou,Wei Li, and Peter J Liu. 2020. Exploring the lim its of transfer learning with a unified text-to-text transformer. Journal of machine learning research, 21(140):1-67.

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

[33] Luciano Del Corro, Allie Del Giorno, Sahaj Agarwal, Bin Yu, Ahmed Awadallah, and Subhabrata Mukher jee. 2023. Skipdecode: Autoregressive skip decoding with batching and caching for efficient llm inference.arXiv preprint arXiv: 2307. 02628.

[34] Shuzhou Yuan, Ercong Nie, Bolei Ma, and Michael Färber. 2024. Why lift so heavy? slimming large language models by cutting off the layers. arXiv preprint arXiv:2402.11700.

[35] Wei Zhu. 2021. Leebert: Learned early exit for bert with cross-level optimization. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th Internationaly Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 2968-2980.

[36] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2019. The curious case of neural text degeneration. arXiv preprint arXiv:1904.09751.

[37] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adap-tation of large language models. arXiv preprint arXiv:2106.09685.

[38] Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. 2022. Lst: Ladder side-tuning for parameter and memoryefficient transfer learning. Advances in Neural Information Processing Systems, 35:12991-13005.

[39] Ronald J Williams. 1992. Simple statistical gradientfollowing algorithms for connectionist reinforcement learning. Machine learning, 8:229-256.

[40] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approxima- tion. Advances in neural information processing systems, 12.

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

[42] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2015. Sequence level train ing with recurrent neural networks. arXiv preprint arXiv:1511.06732.

[43] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platformfor natural language understanding. arXiv preprint arXiv:1804.07461.

[44] 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 con ference on empirical methods in natural language processing: system demonstrations, pages 38-45.

[45] 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.

[46] 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.

[47] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Al shamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hess- low, Julien Launay, Quentin Malartic, et al. 2023. The falcon series of open language models. arXiv preprint arXiv:2311.16867.

[48] Mario Michael Krell, Matej Kosec, Sergio P Perez, and Andrew Fitzgibbon. 2021. Efficient sequence pack- ing without cross-contamination: Accelerating large language models without impacting performance. arXiv preprint arXiv:2107.02027.

[49] Ilya Loshchilov and Frank Hutter. 2017. Decou pled weight decay regularization. arXiv preprint arXiv:1711.05101.

[50] Yifei Yang, Zouying Cao, and Hai Zhao. 2024. Laco: Large language model pruning via layer collapse. arXiv preprint arXiv:2402.11187.

[51] Koyel Chakraborty, Siddhartha Bhattacharyya, and Rajib Bag. 2020. A survey of sentiment analysis from social media data. IEEE Transactions on Computational Social Systems, 7(2): 450-464.

[52] Weiwei Sun, Lingyong Yan, Xinyu Ma, Pengjie Ren, Dawei Yin, and Zhaochun Ren. 2023. Is chatgpt good at search? investigating large lan guage models as re-ranking agent. arXiv preprint arXiv:2304.09542.

[53] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiy- ong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. 2022. A survey on in-context learning. arXiv preprint arXiv: 2301. 00234.

Appendices

A.1 Using LLMs for NLU task

NLU problems can all be viewed as classification problems. Applying LLMs to classification tasks is especially important for applications. First, classification tasks are widely used in many real-life scenarios. For example, conducting sentiment polarity analysis on social media comments, product reviews, etc., helps understand users' attitudes and emotional tendencies towards products, services, or events [51]. Additionally, classifying the relation or similarity of two sentences aids information retrieval [52]. Second, although LLMs are powerful, they require intermediate steps to better complete tasks, such as demonstrations [53] and reasoning prompts [10]. Chae and Davidson [14] uses instruction-following datasets to finetune LLMs for classification tasks. In our work, we fine-tune LLMs with prompts to enhance their performance.

A.2 Prompt design for NLU datasets

Here we provide the prompt we used for some datasets as examples:

Microsoft Research Paraphrase Corpus: Given two sentences, determine if these sentences are semantically equivalent. Output ‘equivalent’ or ‘not equivalent’.

SST-2 (Stanford Sentiment Treebank): Given a sentence, evaluate the sentiment of this sentence as either positive or negative. Output ‘positive’ or ‘negative’.

QQP (Quora Question Pairs): Given two questions, evaluate whether these questions are semantically similar. Output ‘similar’ or ‘not similar’.

A.3 Details of the LLMs used for experiments

We show the information of the large language models we used in Table 4.

Table 4. Information of LLMs.

Model	parameter	hidden size	layer
GPT2-XL [45]	1.5B	1600	4
OPT-1.3B [46]	1.3B	2048	2
Falcon-7B [47]	7B	4544	3
Llama2-7B [32]	7B	4096	3

A.3.1 Results of other LLMs using our method

To validate the generalization ability of our proposed method, we test it on other LLMs on NLU datasets and show the results in Table 5.

Table 5. F1-score of method with different LLMs on NLU tasks.

	RTE		MRPC		CoLA		SST-2	
GPT2-XL	62.5	68.5	77.4	82.5	42.3	46.1	85.4	88.2
OPT-1.3B	63.7	70.5	79.5	85.3	40.7	45.2	86.3	88.4
Falcon-7B	66.8	75.9	84.2	89.1	46.3	55.1	90.2	92.4
Llama-7B-Chat	68.8	77.5	85.6	90.5	49.4	58.1	90.6	92.7