AdaInfer: Instance-aware Adaptive Inference for LLMs

Anonymous ACL submission

Abstract

Large Language Models (LLMs) inference phase is very expensive. An ideal inference-LLM should utilize fewer computational resources while still maintaining its capabilities in generalization and in-context learning ability. In this paper, we try to answer the question, "During LLM inference, can we use shallow layers for easy input; deep layers for hard ones?" To answer this question, we first indicate that *Not all Layers are Necessary at inference time* by statistically analyzing the activated layers across tasks. Then, we propose a simple algorithm named AdaInfer for instance-aware adaptive inference, which determines the inference termination time based on the input instance itself. More importantly, AdaInfer does not alter LLM parameters and maintains generalizability across tasks. Experiments on well-known LLMs (*i.e.,* Llama2-7B/13B and OPT-13B) show that AdaInfer can save 10% to 50% of computational resources on mainstream tasks (*e.g.,* knowledge-based/common-sense QA, text classification). Meanwhile, maintaining accuracy with average minimal (<1%) loss. Additionally, this method is orthogonal to other model acceleration techniques (*e.g.,* sparse and flash attention), offering the potential for further enhancing inference efficiency. Code and data is available at Anomynous Github.

1 Introduction

Large Language Models (LLMs) have demonstrated impressive performance on various downstream tasks (*e.g.,* text generation, summarization, translation, question & answering) using a variety of evaluation protocols such as zero-shot, few-shot, and fine-tuning (Todd et al., 2024; Chan et al., 2022; Kossen et al., 2023; Wang et al., 2023). Notably, their in-context learning ability allows them to adapt to tasks using input-output examples without parameter updates (Kossen et al., 2023; Todd et al., 2024). However, their inference phase is very expensive (Pope et al., 2023; Liu et al., 2023). For example, the inference time complexity for large models like Llama2 (Touvron et al., 2023) is $LSd(d+S)$ per single inference, where d denotes the word vector dimension, S is the sequence length, and L represents the number of layers. An ideal inference LLM should utilize fewer computational resources while still maintaining its capabilities in generalization and in-context learning ability (Liu et al., 2023). The simplest methods for achieving efficient inference in LLMs include model pruning (Liu et al., 2018) and sparse models (LeCun et al., 1989). The potential drawbacks of the aforementioned methods include the following: (i) Few methods consider dynamically reducing the number of activated neurons as an approach to accelerate LLM inference. (ii) Altering LLM parameters may risk compromising its generalization ability, which is challenging to detect. (iii) Different LLM designs pose compatibility challenges with other acceleration methods.

Due to the mentioned issues, inspired by the human thinking process, where quick answers are often provided for simple questions while more time is spent on thoughtful reasoning for knowledge-related questions (Salthouse, 1996; Deary et al., 2001). Existing studies (Teerapittayanon et al., 2016; Huang et al., 2017) show that "Easy" tasks activate at shallower layers while "hard" ones at deeper layers. Motivated by this, we observed that this trend also holds true at LLM inference time, as evidenced by statistics on mainstream LLMs across different tasks. This leads us to hypothesize that *Not all Layers are Necessary at Inference time* with varying input instances (Section 2.2). Therefore, a natural approach to achieve LLM efficient inference for various tasks is adaptive inference based on input instances. This involves selectively executing different network layers for different samples. For instance, allocating fewer computational resources for processing "simple" samples to enhance operational efficiency. This approach trims needless computations for "simpler" inputs, improving efficiency. Additionally, delving into adaptive LLM inference could establish connections between LLMs and the brain's information processing (Hubel and Wiesel, 1962; Murata et al., 2000), facilitating the analysis of activated network modules during sample processing (Han et al., 2021) and determining the critical input components influencing the final prediction.

In this paper, we introduce a simple and straightforward algorithm, named AdaInfer, for instance-aware adaptive inference in LLMs, building on the observation that *Not all Layers are Necessary at Inference time*. The core of instance-aware adaptive inference lies in data-driven decision-making. There are generally two approaches to getting decision-making signals: (1) updating LLM parameters demands training, involves high costs, and might decrease the model's generalizability, and (2) keeping parameters unchanged, a more desirable and cost-effective approach that preserves the model's innate ability (Yao et al., 2023; Zhao et al., 2023). Our proposed AdaInfer decides when to stop inference based on input instance, optimizing efficiency without altering the model's parameters.

AdaInfer keeps LLM parameters unchanged and maintains generalizability across tasks. We adopt the early-exit mechanism from dynamic depth for instance-aware adaptive inference. Specifically, we began by performing statistical analysis on LLM for each block feature (*e.g.,* logits, hidden state, mlp, and attention activation value). Subsequently, we choose logits to construct the feature vector and employ a classic statistical classifier to facilitate the early exit decision-making strategy (see Section 3).

To the best of our knowledge, this is the first attempt to discover that each block's logits are crucial elements for early-exit classifiers in LLMs, and we incorporate it as a fundamental design choice in AdaInfer. Our experiments on well-known LLMs (*i.e.,* OPT-13B and Llama2-7B/13B) demonstrate that AdaInfer, without modifying any model parameters, can save 10% to 50% of computational resources on mainstream tasks (*e.g.,* knowledge-based and common-sense question answering, text classification) while maintaining accuracy with minimal (less than 1%) loss. More importantly, AdaInfer is orthogonal to other model acceleration techniques (*e.g.,* quantization, sparse models, and flash attention), offering the potential for further enhancing inference efficiency (Section 4).

2 LLMs Efficiency at Inference: Not all Layers are Necessary

This section aims to prove that *Not all Layers are Necessary at inference time* by analyzing the number of activated layers across various tasks at LLM inference. We first briefly review LLM's critical components in Section 2.1. Then, we present our statistical observations and insights in Section 2.2.

2.1 Preliminary: LLM Buliding blocks

Modern LLMs are rooted in the Transformer architecture (Vaswani et al., 2017), and can be trained with different unsupervised training objectives. For instance, mainstream LLMs (*e.g.,* GPT, Llama) are pretrained with a full language modeling objective with a decoder-only structure, computing loss on all tokens. The key components of LLMs can be broken down into the following blocks:

Tokenizer and Embedding Layer. This block is responsible for tokenizing input text into individual tokens and transforming them into numerical vectors. These numerical vectors enable the model to process and analyze textual data effectively.

Decoder Block. The decoder block receives numerical vectors, process them through self-attention mechanisms and feedforward neural networks to understand contextual nuances, and outputs predictions for the next token in a sequence.

Classification Layer. Also known as the LM head layer, it transforms decoder logits into a vocabulary-wide probability distribution using linear transformation and softmax, enabling word prediction by selecting the top probability option.

These blocks allow LLMs to efficiently execute tasks like text generation and classification in natural language processing. LLMs employ multi-layer Transformers, focusing much of the computation on decoder blocks. For LLMs like Llama2, inference complexity is $LSd(d+S)$ per single inference, where d is the word vector dimension, S is the sequence length, and L represents the number of decoder blocks. Consequently, to explore the possibility of skipping intermediate layers in LLMs during inference, we do following experiments.

2.2 Not all Layers are Necessary at Inference time

Earlier Transformer models typically comprised 6 decoder layers, while current open-source models, such as Llama2-13B (Touvron et al., 2023),

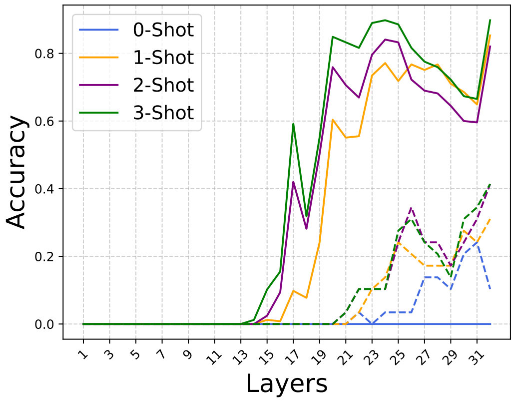


Figure 1: LLama2-7B model performance across 32 Layers: solid line for sentiment analysis and dashed line for MMLU tasks.

feature 40 decoder layers. However, during inference, each input instance for different tasks passes through every block layer by layer until the last layer, prompting us to question: "Can we allocate fewer computational resources per input instance instead of the same substantial budget?" To investigate this, we statistically analyze the number of activated layers across various tasks during inference. Given one task, we examine the relationship between accuracy and the number of layers activated. The statistical results are depicted in Figure 1.

Observation 1: *Not all Layers are Necessary at Inference time: Early Stopping works.* In sentiment analysis using the Llama2-13B (40 layers) model, the average activated layer count per input is 21, with a variance of 5.1. This observation is intuitive. For instance, simpler inputs like "I like Camera A" activate 16 layers, while more complex inputs like "Camera A is better than Camera B in picture quality" activate 24 layers. The latter sentence introduces a comparative sentiment about the "quality" aspect between Camera A and Camera B, which embodies more complex features, suggesting deeper layers for such complex instances.

Observation 2: *Varying Task Difficulties, Different Activation Layers: Stop Simpler Tasks Sooner, Let Complex Ones Go Deeper.* Tasks in the LLM activate different layers, with simpler ones usually at shallower layers and more complex ones at deeper layers. This is shown in Figure 1, which demonstrates the performance of a Llama2-7B model across 32 layers in sentiment analysis (Socher et al., 2013) and MMLU (Hendrycks et al., 2021). For simple tasks like sentiment classification, accuracy matches that of the final layer by the 24th layer. Conversely, for complex tasks like MMLU, accuracy tends to improve with deeper layers.

Insight. The observations mentioned above are intuitive. It's worth noting that similar observations have been made by (Teerapittayanon et al., 2016; Huang et al., 2017) for visual tasks in convolutional neural networks. Surprisingly, we have also observed this phenomenon at LLM inference time. By exploiting this phenomenon, we can perform instance-aware adaptive inference for LLMs, dynamically adjusting their structure/parameters for different test samples, thereby achieving superior advantages in inference efficiency and adaptability. Moving forward, we will leverage this observation to implement adaptive inference.

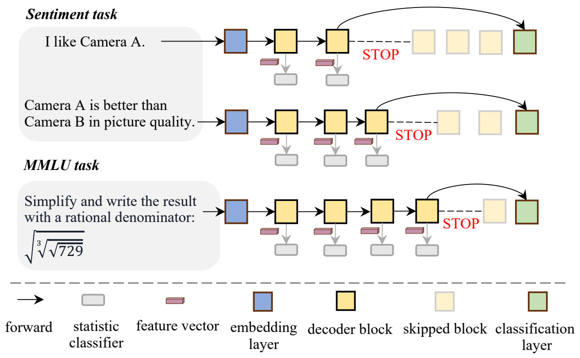
3 AdaInfer

To lower inference computational costs, we introduced AdaInfer, an instance-aware adaptive inference algorithm for LLMs. The core of instance-aware adaptive inference lies in data-driven decision-making fortified by integrating the early exit strategy from dynamic depth. AdaInfer dynamically computes the stopping signal by evaluating critical features (*i.e.,* "ga" and "top prob"). AdaInfer consists of a Feature Selection module and a Classifier module, with SVM or CRF being common classifier choices. At each layer, Feature Selection crafts a feature vector for the current instance. Then Classifier determines if the stopping signal is strong enough for an early stop. If a stop signal is confirmed, we can discard the remaining layers. Figures 2a and 2b visually depict AdaInfer's workflow and the computational efficiencies gained through this method, respectively.

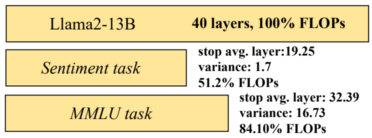
3.1 Feature Selection

In the LLM era, there are two typical approaches to obtaining the decision-making signal. The first approach involves modifying LLM parameters, which requires training and incurs high costs. More importantly, it may pose a potential risk of compromising the model's generalization capabilities in other aspects and its in-context learning abilities, and detecting these issues can be challenging (Gu et al., 2024; Yao et al., 2023).

Hence, we embrace a more desirable and cost-effective approach that preserves the model's innate abilities without altering parameters. This approach is enhanced by an early-exit mechanism



(a) An illustration of AdaInfer processing three input instances, including two for sentiment analysis and one for a knowledge-based question answering task. It shows that the early-exit times differ among the instances.



(b) After implementing AdaInfer, LLMs can reduce computational costs through adaptive early-exit strategies.

Figure 2: An illustration of AdaInfer's processing and computational savings.

for efficiency. AdaInfer utilizes specially designed features such as "gap" and "top prob", leveraging a statistical classifier for evaluation to generate the stopping signal. The rationale behind selecting these specific features is explained further on.

Problem: The lack of universal-level features. LLMs capture coarse-grained features in their initial layers and develop more detailed, fine-grained representations in subsequent, deeper layers, facilitated by repeated application of multi-head attention mechanisms and the use of residual connections. However, there is a lack of universal-level features to demonstrate that shallow-level representation is sufficient for the current task. Furthermore, these features need to be inherently universal to ensure compatibility across various LLMs.

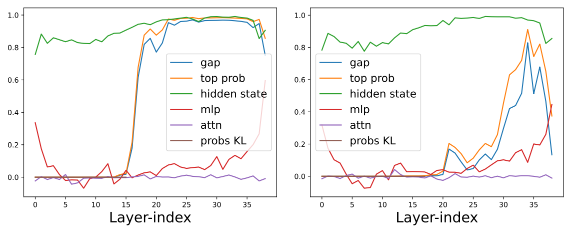
Solution: Logits reflect mutation. To address this, we conducted a visual analysis of diverse features across the layers within each block of LLMs. Our examination focused specifically on:

- **gap**: Measures the current block's prediction confidence for the next token, defined as gap $= |P(\text{top token}) - P(\text{second token})|$, where P represents the probability distribution generated by the current block.

- **top prob**: Indicates $P(\text{top token})$, the probability estimation by the current block for the most likely next token.

- **cosine similarity**: Calculated to evaluate the similarity between the features of current and previous block, including attention activation value (attn), multi-layer perceptron outputs (mlp), and hidden states.

These analyses are showcased in Figure 3. In this figure, we observe the following trends: (1) For Llama2 (13B, 40 layers) (Touvron et al., 2023) across sentiment and MMLU tasks, the gap and top prob gradually increase during the inference phase, stabilizing in the deeper layers. (2) The activation of gap and top prob varies across layers for different tasks. These phenomenons are also evident in the Llama2-7B (Touvron et al., 2023), OPT-13B (Zhang et al., 2022), and GPT-J (Wang and Komatsuzaki, 2021) (See Appendix D). This demonstrates gap and top prob can serve as universal features, indicating the stopping signal. Notably, gap and top probability values remain consistent across diverse tasks, suggesting a versatile classifier applicable to various tasks. We also conduct factor study in subsequent experiments to show that other features exhibit subtle differences across layers.



(a) Llama2 on sentiment (b) Llama2 on MMLU

Figure 3: Statistics of features within LLMs that vary with the forward layer.

3.2 Classifier

Problem: A trade-off between performance and computational efficiency. Using confidence-based criteria doesn't require extra computations during inference but may involve threshold tuning with validation data to balance accuracy and efficiency (Huang et al., 2017; Yang et al., 2020). Conversely, the plug-and-play nature of the gating function (Lin et al., 2017; Bejnordi et al., 2019) provides greater universality. Nonetheless, discrete decision functions, lacking gradient information, often require specialized training methods.

Solution: Connect Block Features to Decision Making via Statistical Classifier. Considering the trend in Figure 3, classical statistical classification methods can address issues with discrete decision functions. By classifying general features like gap and top prob, we simplify decision-making into binary classification, enabling early exit decisions. If the classifier deems the current layer's features stoppable, subsequent layers can halt computation; otherwise, they proceed to the final layer. This process is also illustrated in Figure 2a.

3.3 Classifier Objective

Given one instance, we calculate the feature vector x_d using the feature selection module. This feature vector serves as the input for the classifier module. If the current layer's output $\hat{y}$ provides the correct answer y, the associated label y_c is a positive example; otherwise, it's a negative example.

$$y_c = \begin{cases} 1 & \text{if } \hat{y} = y, \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

Thus, for an $L-$layer LLM, each input instance x yields L pairs of $< x^d, y_c >$. The details of creating training data for classifier are in Appendix C. We consider two types of classifiers, Support Vector Machines (SVM) (Hearst et al., 1998) and Conditional Random Fields (CRF) (Lafferty et al., 2001). The first one does not rely on the context of sequences, while the second one takes into account that the features of layer-by-layer blocks might implicitly incorporate concepts of sequence modeling.

SVM objective. SVM aims to find an optimal hyperplane that separates classes by minimizing classification errors and maximizing the margin between support vectors.

CRF objective. CRF is used to capture sequence feature dependencies and make decisions based on neighboring element states in sequence labeling tasks, with the training objective of maximizing the conditional likelihood of the true label sequence given the input sequence.

Table 1: LLMs statistics using AdaInfer.

Model	Params	Tokens	$\|L\|$
Meta/OPT	13B	180B	40
Meta/Llama 2	7B	2T	32
Meta/Llama 2	13B	2T	40
Meta/Llama 2	70B	2T	80

4 Experiments

Building on the hypothesis *Not all Layers are Necessary during LLM inference* (Section 2.2), we conduct experiments with algorithm AdaInfer on well-known LLMs across various tasks.

4.1 Evaluation Tasks

To evaluate the zero/few shot learning capabilities of AdaInfer, we utilize three primary types of tasks.

Question Answering Task Group. (1) MMLU (Hendrycks et al., 2021) encompasses 57 tasks across humanities, social sciences, STEM, and more, requiring world knowledge and problem-solving capabilities. (2) CommonsenseQA (Talmor et al., 2019) tests for commonsense knowledge through multiple-choice questions. (3) SQuAD (Rajpurkar et al., 2016) serves as a reading comprehension benchmark, with questions based on Wikipedia articles and answers are either segments of passage or marked as unanswerable.

Text Classification Task Group. (1) SST-2 (Socher et al., 2013) involves sentiment analysis of movie reviews with binary "positive" or "negative" labels. (2) AG News (Zhang et al., 2015) classifies news headlines and article sentences into Business, Science/Technology, Sports, and World categories.

Rule Understanding Task. GPT-3's (Brown et al., 2020) few-shot learning capability is tested with tasks requiring pattern recognition, using synthetic datasets from (Todd et al., 2024; Hernandez et al., 2024) for tasks like Capitalize/Lowercase Letter, Choose Item/Category from List, and recognizing data pairs (*e.g.*,, Person-Occupation).

4.2 Experiment Settings

Large Language Models. For AdaInfer's backbone, we chose widely recognized LLMs, detailed in Table 1. Our selections encompass OPT (Zhang et al., 2022) and the Llama 2 series (Touvron et al., 2023), which display nuanced variations in architectural design and training data size.

In-context Learning setting. We evaluate our approach under zero-shot and few-shot scenarios, using sample sizes of 5, 10, 15, and 20. For zero-shot, the input is the test set's x_q. For few-shot, training set examples are added to x_q. For in-context learning prompts, we use a default template: Q : $\{x_k\}$ \nA : $\{y_k\}$ \n\n, concatenating random x_k and y_k samples from task-specific training sets.

Table 2: Performance and Efficiency in Question Answering Tasks, with Accuracy (%) denoted by 'acc'. Results include Few-shot learning with sample sizes of 5, 10, 15, and 20, showcasing the average values.

Setting	Model	MMLU		CommonsenseQA		SQuAD		Avg	
		Acc↑	FLOPs↓	Acc↑	FLOPs↓	Acc↑	FLOPs↓	Acc↑	FLOPs↓
Zero-shot	OPT-13B	7.95	100	8.20	100	20.00	100	12.05	100
	AdaInfer	8.67	97.55	2.80	97.55	23.00	97.55	11.49	97.55
Few-shot	OPT-13B	23.60	100	21.45	100	26.12	100	23.72	100
	AdaInfer	22.59	83.94	21.62	86.05	25.95	88.31	23.39	86.10
Zero-shot	Llama2-13B	2.54	100	1.00	100	19.20	100	7.58	100
	AdaInfer	2.48	98.14	0.70	98.37	25.90	85.34	9.69	93.95
Few-shot	Llama2-13B	53.31	100	64.92	100	52.9	100	57.04	100
	AdaInfer	52.44	93.55	62.48	89.10	48.35	80.66	54.42	87.77

Table 3: Performance and Efficiency in classification and rule understanding, with Accuracy (%) denoted by 'acc'. Results include Few-shot learning with sample sizes of 5, 10, 15, and 20, showcasing the average values.

Setting	Model	Sentiment		AG News		Avg		Rule Understanding	
		Acc↑	FLOPs↓	Acc ↑	FLOPs↓	Acc↑	FLOPs↓	Acc↑	FLOPs↓
Zero-shot	OPT-13B	0.00	100	0.10	100	0.05	100	3.38	100
	AdaInfer	0.00	96.87	0.10	100	0.05	98.44	3.86	92.52
Few-shot	OPT-13B	92.58	100	72.83	100	82.71	100	58.48	100
	AdaInfer	92.97	80.28	72.83	100	82.90	90.14	52.83	89.74
Zero-shot	Llama2-13B	0.00	100	0.10	100	0.05	100	2.32	100
	AdaInfer	0.00	97.43	0.10	88.37	0.05	92.90	6.14	85.76
Few-shot	Llama2-13B	95.90	100	77.53	100	86.72	100	69.36	100
	AdaInfer	92.65	59.70	76.43	87.69	84.54	73.70	61.87	80.61

Metric. For performance evaluation, we report the top-1 accuracy score on the test set following (Todd et al., 2024). To assess computational cost, we determine the early exit layer index for each input instance, which can be translated into floating-point operations (FLOPs) ratios for comparison using the method described in (Narayanan et al., 2021). The FLOPs ratio is calculated as:

$$\frac{2l'(6h+s)+V}{2l(6h+s)+V} \tag{2}$$

Where l' represents the stop layer index during inference in AdaInfer, l is the total number of transformer layers, h denotes the hidden size, s is the sequence length, and V stands for vocabulary size. Further details on the calculation process can be found in Appendix B. Since statistical classifiers entail significantly lower computational costs compared to LLM inference, as detailed in Appendix B, we can overlook this aspect in our analysis.

4.3 AdaInfer: Comparable Performance with Lower Computational Costs

The main experimental results of AdaInfer are presented in Tables 2 and 3. These experiments were conducted in zero-shot and few-shot settings, showcasing the Top-1 accuracy and average FLOPs ratios (compared to the baseline). From a perspective of performance and computational efficiency, we can draw the following experimental conclusions.

Performance is Comparable with Minimal Loss. Tables 2 and 3 show that across both zero-shot and few-shot settings, top-1 average accuracy remains within a narrow margin of <5% for all tasks and <1% for QA and text classification task groups, when compared to baseline models. AdaInfer maintains mainstream LLM capabilities and in-context learning abilities without modifying model parameters. This finding is promising, especially in light of our observation1 in Section 2.2, where we demonstrate the feasibility of implementing early exit strategies within LLM middle layers while preserving accuracy. For certain tasks, AdaInfer surpasses the last layer (baseline) in zero-shot or few-shot accuracy. This hints at a tendency for deep layers to potentially over-represent certain tasks, which could impede performance during LLM inference.

Reducing FLOPs Savings from 10% to 40%. We convert the average and variance of early exit layers for each task to FLOPs ratios in Table 2 and Table 3. It can be observed that the FLOPs ratios vary for different types of tasks, ranging from 90% to 60%. This variation is due to the fact that AdaInfer assesses different early exit layer configurations for different task inputs. Even for the same task with different inputs, AdaInfer may recommend different early exit layer settings. For instance, in the sentiment analysis task, a 40% reduction in computational cost can be achieved using Llama2-13B, while for the knowledge-based question answering MMLU and Commonsense question answering CommonSenseQA, the savings range from 10% to 20%. This aligns with our observation2 outlined in Section 2.2, where we argue that at LLM inference scenario, *Not all Layers are Necessary*, and allocating fewer computational resources for "simple" samples can improve computational efficiency.

Table 4: Comparative Analysis of GAP and CRF on Performance and Computational Efficiency.

Task	Setting	AdaInfer *w.* GAP		AdaInfer *w.* CRF	
		Acc↑	FLOPs↓	Acc↑	FLOPs↓
MMLU	Zero-shot	5.35	90.84	4.77	97.40
	Few-shot	47.09	84.10	52.72	97.15
CommonsenseQA	Zero-shot	1.10	92.78	1.40	97.28
	Few-shot	55.33	79.57	65.72	96.40
SQuAD	Zero-shot	24.60	73.17	23.10	93.03
	Few-shot	43.43	71.19	51.75	89.94
Sentiment	Zero-shot	0.00	88.25	0.00	97.27
	Few-shot	91.45	51.25	95.60	73.07
AG News	Zero-shot	0.10	77.82	0.10	94.04
	Few-shot	69.17	70.65	76.77	93.08
Rule Understanding	Zero-shot	9.90	74.80	3.43	90.29
	Few-shot	53.78	70.38	65.82	90.29

4.4 GAP vs. CRF: A Comparative Study

In the main experiments Table 2 and Table 3, we employed SVM as the classifier for AdaInfer. To explore the impact of different classification strategies, Table 4 compares the effects of implementing an early-exit strategy with a GAP threshold set at 0.8 (stopping computation when the current block's GAP feature exceeds 0.8) against using CRF as a classifier. The results indicate that both GAP and CRF can reduce computational costs without sacrificing LLM performance. Notably, in the zero-shot setting, GAP outperforms CRF, suggesting a relatively weak dependency between block features.

4.5 AdaInfer Performance Across Scales: 7B and 70B Insights

In our main experiments, we employed 13B-sized Llama and OPT models. To explore the effects of AdaInfer on models of different sizes, we conducted experiments on the 7B and 70B versions of Llama. The results for the 7B model, presented in Table 5, show that AdaInfer either maintains accuracy with minimal (<1%) loss or exceeds the baseline in certain tasks, and achieves a computational reduction ranging from 10% to 50%. However, in experiments with the 70B model, we observed that in a zero-shot setting, AdaInfer matches or slightly exceeds the baseline model while reducing computational costs by 30% to 60%. Notably, in the few-shot setting, despite similar reductions in computation, AdaInfer's accuracy shows a 1% to 25% gap across different tasks compared to the baseline. This suggests that for larger models, such as the 70B or even larger scales, AdaInfer may need to more precisely identify and utilize features at different levels. Improving AdaInfer to adapt to these larger models is a direction for our future research.

Table 5: AdaInfer on Llama2-7B Across Tasks for Performance and Computational Efficiency.

Task	Setting	Llama2-7B		AdaInfer	
		Acc↑	FLOPs↓	Acc↑	FLOPs↓
MMLU	Zero-shot	4.19	100	4.63	96.13
	Few-shot	43.05	100	43.73	93.76
CommonsenseQA	Zero-shot	5.30	100	4.80	95.26
	Few-shot	53.50	100	53.00	90.46
SQuAD	Zero-shot	20.40	100	23.80	89.98
	Few-shot	48.08	100	45.82	87.06
Sentiment	Zero-shot	0.00	100	0.00	96.37
	Few-shot	95.20	100	95.30	68.05
AG News	Zero-shot	0.10	100	0.10	91.36
	Few-shot	79.65	100	79.72	94.51
Rule Understanding	Zero-shot	5.47	100	5.32	91.55
	Few-shot	66.80	100	66.92	88.41

The results of all LLMs using different classifiers are summarized in Table 8 and Table 9 in the Appendix E and we have highlighted the best results for each task in the current setting.

4.6 Generalization effect on Mainstream Classifier

In Table 2 and Table 3, we randomly selected 6 to 9 training datasets from all task training sets (note:

Table 6: Generalization Performance of statistic classifier on Sentiment Task on Llama2-7B (32 Layers), Inter-Model refer to Llama2-13B(40 layers).

Classifier	Generalization	Acc	Layers	Varience	FLOPs
SVM	Intra-Task	94.90	18.15	0.45	60.58
CRF		0.00	0.00	0.00	100
SVM	Inter-Task	95.5	19.2	4.40	63.80
CRF		94.9	20.2	4.55	66.87
SVM	Inter-Model	90.70	20.60	3.70	54.55
CRF		87.75	19.20	2.75	51.09

there are 71 subdatasets in total). Furthermore, to assess the generalization performance of the statistical classifiers, we conducted the following tests.

- **Intra-Task Generalization.** Evaluating the sentiment task using a classifier trained on the sentiment training dataset.
- **Inter-Task Generalization.** Testing sentiment using a classifier trained on the knowledge question-answering task's dataset.
- **Inter-Model Generalization.** Assessing the sentiment task on Llama2-13B using a classifier trained on Llama2-7B.

The results are presented in Table 6. The SVM classifier exhibits satisfactory intra-task and inter-task generalization capabilities, consistent with the results presented in the main results. However, for the CRF classifier, training in an intra-task manner leads to premature termination of the LLM at very shallow layers, resulting in subpar performance. This could be attributed to insufficient feature selection, causing the CRF to overfit noise or local features in the training data. Additionally, due to variations in the logits distribution characteristics among different models, the inter-model classifier's performance shows moderate accuracy. In conclusion, based on the results from Table 2 and Table 3 and Table 6, when using AdaInfer, we recommend utilizing SVM as the classifier.

4.7 Factor Study

In response to the features mentioned in Section 3.1, we conducted cross-validation. Given that the classifiers in the main results utilized basic features (*i.e.,* gap, top prob), we explored the impact of features such as the cosine similarity between the current block and the previous block, which encompasses the attention values (attn), multi-layer perceptron (mlp), and hidden states. The experimental results are presented in Table 7. It is evident that attn has no discernible impact on the results, while other features like mlp and hidden states have an adverse effect. This result is consistent with the trend shown in Figure 3, indicating that logits can measure whether the model's current forward progress is sufficient, while changes in attention, hidden state, and mlp involve various factors.

Table 7: Comparative Analysis of SVM Performance with Incremental Feature Addition in Sentiment and MMLU/Anatomy Tasks.

Feature	Sentiment	MMLU
Base Features (gap, top prob)	94.90	41.13
+attn	94.90	41.13
+hidden state	67.53	41.13
+mlp	67.88	41.93

5 Related Work

For decades, various systems have been developed to enhance machine learning model inference efficiency, incorporating techniques such as instance-wise dynamic networks (Han et al., 2021; Huang et al., 2017; Teerapittayanon et al., 2016; Bolukbasi et al., 2017) and quantization (Xiao et al., 2023; Frantar et al., 2022), sparsity (Liu et al., 2023) or distillation (Touvron et al., 2021). They are orthogonal areas and usually excel in different settings. Our work AdaInfer is aligns more closely dynamic networks with further elaboration in Appendix A.

6 Conclusion

In this paper, we first hypothesize that *Not all Layers are Necessary at inference time* and provide statistical evidence to support this. Building on this hypothesis, we present AdaInfer, a simple and straightforward algorithm. This algorithm determines the appropriate moment to cease inference based on the input instance, thus enhancing inference efficiency and adaptability without modifying the model's parameters. Experiments on mainstream LLMs show that AdaInfer can reduce computational usage by 10% to 50% across various tasks all while maintain comparable performance with minimal loss. More importantly, AdaInfer is compatible with other model acceleration techniques, potentially offering further improvements in inference efficiency. We argue that AdaInfer, as a simple yet effective algorithm, establishes a new paradigm for efficient inference in LLMs.

Limitations

In this paper, we make a first attempt to discover that the logits of each block are critical for early-exit classifiers in LLMs, incorporating this insight as a key design choice in AdaInfer. However, since AdaInfer relies on a single forward pass, it has not yet been extended to sequential generative tasks , offering significant avenues for future research.

Ethics Statement

Our research aims to optimize large-scale model inference without modifying parameters, promising efficiency gains and reduced energy consumption. However, we must address potential misuse concerns, as enhanced inference capabilities may also enable malicious actors to exploit large neural language systems by injecting or amplifying logits as features, leading to undesirable behavior.

References

Mikel Artetxe, Shruti Bhosale, Naman Goyal, Todor Mihaylov, Myle Ott, Sam Shleifer, Xi Victoria Lin, Jingfei Du, Srinivasan Iyer, Ramakanth Pasunuru, et al. 2021. Efficient large scale language modeling with mixtures of experts. arXiv preprint arXiv:2112.10684.

Babak Ehteshami Bejnordi, Tijmen Blankevoort, and Max Welling. 2019. Batch-shaping for learning conditional channel gated networks. arXiv preprint arXiv:1907.06627.

Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. 2017. Adaptive neural networks for efficient inference. In International Conference on Machine Learning, pages 527–536. PMLR.

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.

Stephanie Chan, Adam Santoro, Andrew Lampinen, Jane Wang, Aaditya Singh, Pierre Richemond, James McClelland, and Felix Hill. 2022. Data distributional properties drive emergent in-context learning in transformers. Advances in Neural Information Processing Systems, 35:18878–18891.

Ian J Deary, Geoff Der, and Graeme Ford. 2001. Reaction times and intelligence differences: A population-based cohort study. Intelligence, 29(5):389–399.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. The Journal of Machine Learning Research, 23(1):5232–5270.

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.

Jia-Chen Gu, Hao-Xiang Xu, Jun-Yu Ma, Pan Lu, Zhen-Hua Ling, Kai-Wei Chang, and Nanyun Peng. 2024. Model editing can hurt general abilities of large language models. arXiv preprint arXiv:2401.04700.

Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. 2021. Dynamic neural networks: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 44(11):7436–7456.

Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. 1998. Support vector machines. IEEE Intelligent Systems and their applications, 13(4):18–28.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. Proceedings of the International Conference on Learning Representations (ICLR).

Evan Hernandez, Arnab Sen Sharma, Tal Haklay, Kevin Meng, Martin Wattenberg, Jacob Andreas, Yonatan Belinkov, and David Bau. 2024. Linearity of relation decoding in transformer language models. In Proceedings of the 2024 International Conference on Learning Representations.

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.

Weizhe Hua, Yuan Zhou, Christopher M De Sa, Zhiru Zhang, and G Edward Suh. 2019. Channel gating neural networks. Advances in Neural Information Processing Systems, 32.

Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Multi-scale dense networks for resource efficient image classification. arXiv preprint arXiv:1703.09844.

David H Hubel and Torsten N Wiesel. 1962. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. The Journal of physiology, 160(1):106.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. 1991. Adaptive mixtures of local experts. Neural computation, 3(1):79–87.

Jannik Kossen, Tom Rainforth, and Yarin Gal. 2023. In-context learning in large language models learns label relationships but is not conventional learning. arXiv preprint arXiv:2307.12375.

John Lafferty, Andrew McCallum, and Fernando CN Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

Yann LeCun, John Denker, and Sara Solla. 1989. Optimal brain damage. Advances in neural information processing systems, 2.

Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. 2017. Runtime neural pruning. Advances in neural information processing systems, 30.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2018. Rethinking the value of network pruning. arXiv preprint arXiv:1810.05270.

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

Akira Murata, Vittorio Gallese, Giuseppe Luppino, Masakazu Kaseda, and Hideo Sakata. 2000. Selectivity for the shape, size, and orientation of objects for grasping in neurons of monkey parietal area aip. Journal of neurophysiology, 83(5):2580–2601.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. 2021. Efficient large-scale language model training on GPU clusters using megatron-lm. In International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2021, St. Louis, Missouri, USA, November 14-19, 2021, page 58. ACM.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. Proceedings of Machine Learning and Systems, 5.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. arXiv e-prints, page arXiv:1606.05250.

Timothy A Salthouse. 1996. The processing-speed theory of adult age differences in cognition. Psychological review, 103(3):403.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4149–4158, Minneapolis, Minnesota. Association for Computational Linguistics.

Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. 2019. Distilling task-specific knowledge from bert into simple neural networks. arXiv preprint arXiv:1903.12136.

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

Eric Todd, Millicent L. Li, Arnab Sen Sharma, Aaron Mueller, Byron C. Wallace, and David Bau. 2024. Function vectors in large language models. In Proceedings of the 2024 International Conference on Learning Representations.

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

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pages 5998–6008.

Paul Viola and Michael J Jones. 2004. Robust real-time face detection. International journal of computer vision, 57:137–154.

Ben Wang and Aran Komatsuzaki. 2021. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax.

Lean Wang, Lei Li, Damai Dai, Deli Chen, Hao Zhou, Fandong Meng, Jie Zhou, and Xu Sun. 2023. Label words are anchors: An information flow perspective for understanding in-context learning. arXiv preprint arXiv:2305.14160.

Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. 2018. Skipnet: Learning dynamic routing in convolutional networks. In Proceedings of the European Conference on Computer Vision (ECCV), pages 409–424.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In International Conference on Machine Learning, pages 38087–38099. PMLR.

Xingrun Xing, Li Du, Xinyuan Wang, Xianlin Zeng, Yequan Wang, Zheng Zhang, and Jiajun Zhang. 2023. Bipft: Binary pre-trained foundation transformer with low-rank estimation of binarization residual polynomials. arXiv preprint arXiv:2312.08937.

Le Yang, Yizeng Han, Xi Chen, Shiji Song, Jifeng Dai, and Gao Huang. 2020. Resolution adaptive networks for efficient inference. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 2369–2378.

Yunzhi Yao, Peng Wang, Bozhong Tian, Siyuan Cheng, Zhoubo Li, Shumin Deng, Huajun Chen, and Ningyu Zhang. 2023. Editing large language models: Problems, methods, and opportunities. arXiv preprint arXiv:2305.13172.

Dewen Zeng, Nan Du, Tao Wang, Yuanzhong Xu, Tao Lei, Zhifeng Chen, and Claire Cui. 2023. Learning to skip for language modeling. arXiv preprint arXiv:2311.15436.

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.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. Advances in neural information processing systems, 28.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2023. Expel: Llm agents are experiential learners. arXiv preprint arXiv:2308.10144.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, Quoc V Le, James Laudon, et al. 2022. Mixture-of-experts with expert choice routing. Advances in Neural Information Processing Systems, 35:7103–7114.

A Related Work

Dynamic Inference. A straightforward approach to address the hypothesis outlined in Section 2 involves the utilization of dynamic neural networks (Han et al., 2021). The core idea of dynamic networks—adaptive inference—has been explored by some researchers before the prevalence of deep networks today (Jacobs et al., 1991; Viola and Jones, 2004). In the context of LLM inference, LLMs dynamically adjusts its own structure and parameters when dealing with diverse test instance. Popular deep network architectures often encompass two dimensions: depth (number of network layers) and width (number of channels, parallel subnetworks, *etc.*). Consequently, networks with dynamic structures can be classified into two types: dynamic depth and dynamic width. Dynamic width in the context of Mixture of Experts (MOE) (Fedus et al., 2022; Zhou et al., 2022; Artetxe et al., 2021), CNN channel pruning (Hua et al., 2019; Hoefler et al., 2021), when applied to the LLM inference scenario, poses a potential risk. This risk is associated with the possibility that modifying model parameters may compromise the model's generalization and detecting these issues can be challenging. Dynamic depth techniques often encompass early-exit and skip-layer mechanisms. The core idea of the early-exit strategy is to set exits at intermediate layers of the model and adaptively decide whether a sample should exit early based on its output at these mid-exits. Inspired by works that integrate classifiers into intermediate layers of CNN/DNN networks for visual tasks to handle early exits (Bolukbasi et al., 2017; Huang et al., 2017; Teerapittayanon et al., 2016), our proposed AdaInfer closely aligns with this concept. The early-exit mechanism effectively bypasses the computation of all layers following a certain classifier. Meanwhile, skip-layer functionality dynamically omits the execution of a layer (or module) for any input token, facilitated by a gate function (Wang et al., 2018) or a binary router (Zeng et al., 2023).

Quantization, Sparsity, Distillation for LLM inference. For decades, various system adaptations for enhancing model inference in machine learning have been under continuous exploration. Among these, techniques like quantization (Xiao et al., 2023; Frantar et al., 2022; Xing et al., 2023) have been employed to accelerate inference by representing floating-point data with fewer bits, thus reducing memory usage during inference. Sparsity (Liu et al., 2023; Frantar and Alistarh, 2023) employs predictors or sparse regression solvers to facilitate sparsity in LLM inference, further optimizing performance. Distillation (Touvron et al., 2021; Tang et al., 2019) reduces computational and storage costs by training a smaller model to mimic the performance of the original model, optimizing efficiency.

B Computation cost.

Classifier Computation cost. We utilized the sklearn library for training SVM[*] and CRF[†], adhering to default configurations. For SVM and CRF training, we used the sklearn library with default settings. Given a training dataset with N training examples, the time complexity for SVM training typically ranges from $O(N^2 \times d)$ to $O(N^3 \times d)$, where d is the feature dimension. SVM prediction time complexity is $O(d)$ per single inference. For standard linear-chain CRF, the training time complexity is approximately $O(N \times S \times M)$, where S is the average sequence length, M is the label count. The prediction time complexity for CRF is $O(S \times M)$ per single inference. In contrast, the inference time complexity for large models like llama2 is $LSd(d + S)$ per single inference, where d is the word vector dimension, S is the sequence length, and L represents the number of layers. Comparatively, the computational load of SVM and CRF is negligible when compared to large models.

Transformer Computation cost. Given a language model with l transformer layers, hidden size h, sequence length s, vocabulary size V, and batch size B. Each transformer block needs $24Bsh^2 + 4Bs^2h$ FLOPs for the forward pass. The other main contributor to the FLOPs count is the classification layer in the language model head, which transforms features of dimension h to the vocabulary dimension V. The required FLOPs for this operation is $2BshV$ in the forward pass. Summing these together, a transformer model with l transformer layers, the total number of floating-point operations for inference is $4Bshl(6h + s) + 2BshV$. Thus, the ratio of inference cost in FLOPs is

$$\frac{2l'(6h + s) + V}{2l(6h + s) + V} \tag{3}$$

where l' is the AdaInfer stop layer index during inference.

[*]https://scikit-learn.org/stable/modules/svm.html
[†]https://sklearn-crfsuite.readthedocs.io/en/latest/

C Details of Creating Training Data for Classifier

Considering a training input instance x and its corresponding label y from D_{train}. Once x is processed through a decoder layer of LLM, we can extract a general feature vector x^d (d is the number of features). Additionally, we obtain the probability distribution P over the vocabulary V of the current layer's hidden state after passing through the classification layer (as depicted in Section 2.1). This can be represented as: $P = softmax(WH + b)$, where H is the hidden state of the current layer, W and b are the weights and bias of the classification layer, respectively. $softmax$ is the softmax function applied to convert logits to probabilities. Let the highest-ranked token in this distribution be denoted as $\hat{y} = \text{argmax}(P)$, where $\text{argmax}(P)$ finds the token with the highest probability. If $\hat{y}$ matches the label y, the associated label y_c for the feature vector x_d is designated as positive; otherwise, it is labeled as negative.

$$y_c = \begin{cases} 1 & \text{if } \hat{y} = y, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

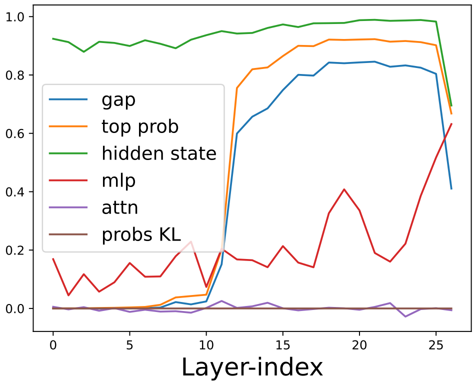
Thus, for an $L-$layer LLM, each input instance x yields L pairs of $< x^d, y_c >$.

D Observation of other LLMs

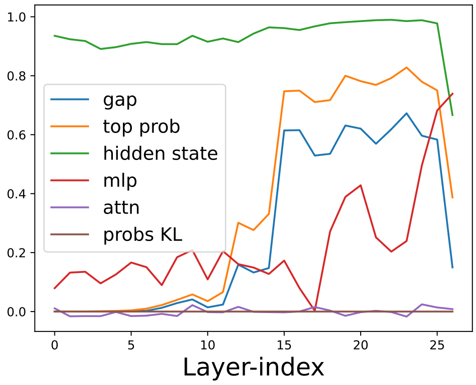
Figure 4 represents visual analysis of diverse features across the layers within each block of mainstream LLMs.

E Comprehensive Summary of Results

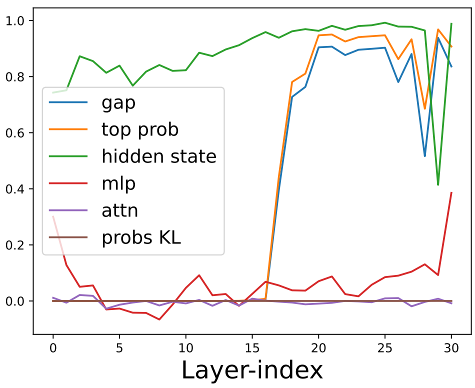
The results of all LLMs using different classifiers are summarized in Table 8 and 9. We have highlighted the best results for each task in the current setting.



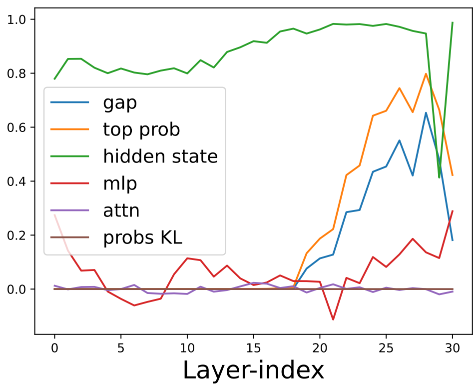
(a) GPT-J 6B on sentiment



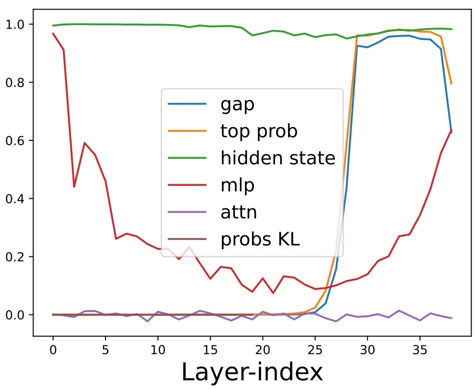
(b) GPT-J 6B on MMLU



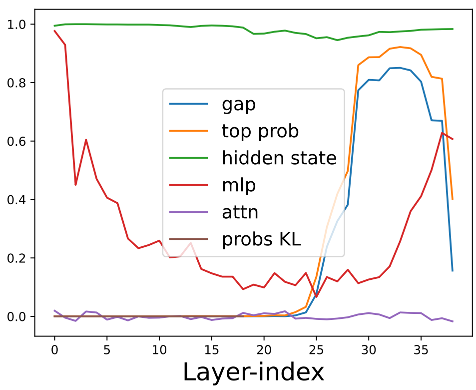
(c) Llama2-7B on sentiment



(d) Llama2-7B on MMLU



(e) OPT-13B on sentiment



(f) OPT-13B on MMLU

Figure 4: visual analysis of diverse features across mainstream LLMs.

Table 8: Performance and Computational Efficiency in Question Answering Tasks, with Accuracy (%) denoted by 'acc'. Results include Few-shot learning with sample sizes of 5, 10, 15, and 20, showcasing the average values.

Setting	Model	MMLU		CommonsenseQA		SQuAD		Avg	
		Acc↑	FLOPs↓	Acc↑	FLOPs↓	Acc↑	FLOPs↓	Acc↑	FLOPs↓
Zero-shot	OPT-13B	7.95	100	8.20	100	20.00	100	12.05	100
	AdaInfer *w.* GAP	3.21	89.58	0.60	85.17	20.72	87.98	8.18	87.58
	AdaInfer *w.* CRF	7.14	96.57	4.60	93.26	24.36	93.22	12.03	94.35
	AdaInfer	8.67	97.55	2.80	97.55	23.00	97.55	11.49	97.55
Few-shot	OPT-13B	23.60	100	21.45	100	26.12	100	23.72	100
	AdaInfer *w.* GAP	20.99	79.54	20.72	80.00	24.20	82.93	21.97	80.82
	AdaInfer *w.* CRF	24.44	97.43	21.18	97.55	25.98	97.11	24.81	97.37
	AdaInfer	22.59	83.94	21.62	86.05	25.95	88.31	23.39	86.10
Zero-shot	Llama2-7B	4.19	100	5.30	100	20.40	100	9.96	100
	AdaInfer *w.* GAP	4.69	95.69	4.60	94.90	23.90	89.48	11.06	93.36
	AdaInfer *w.* CRF	4.86	95.32	2.00	95.01	18.80	91.17	8.55	93.83
	AdaInfer	4.63	96.13	4.80	95.26	23.80	89.98	11.08	93.79
Few-shot	Llama-2-7B	43.05	100	53.50	100	48.08	100	48.21	100
	AdaInfer *w.* GAP	44.03	93.69	52.83	90.23	45.68	86.72	47.51	90.21
	AdaInfer *w.* CRF	41.38	94.23	53.6	91.61	43.62	88.10	46.20	91.31
	AdaInfer	43.73	93.76	53.00	90.46	45.82	87.06	47.52	90.43
Zero-shot	Llama2-13B	2.54	100	1.00	100	19.20	100	7.58	100
	AdaInfer *w.*GAP	5.35	90.84	1.10	92.78	24.60	73.17	10.35	85.60
	AdaInfer *w.*CRF	4.77	97.40	1.40	97.28	23.10	93.03	9.76	95.90
	AdaInfer	2.48	98.14	0.70	98.37	25.90	85.34	9.69	93.95
Few-shot	Llama-2-13B	53.31	100	64.92	100	52.9	100	57.04	100
	AdaInfer *w.* GAP	47.09	84.10	55.33	79.57	43.43	71.19	48.62	78.29
	AdaInfer *w.*CRF	52.72	97.15	65.72	96.40	51.75	89.94	56.73	94.50
	AdaInfer	52.44	93.55	62.48	89.10	48.35	80.66	54.42	87.77

Table 9: Performance and Computational Efficiency in text classification and rule understanding tasks, with Accuracy (%) denoted by 'acc'. Results include Few-shot learning with sample sizes of 5, 10, 15, and 20, showcasing the average values.

Setting	Model	Sentiment		AG News		Avg		Rule Understanding	
		Acc↑	FLOPs↓	Acc ↑	FLOPs↓	Acc↑	FLOPs↓	Acc↑	FLOPs↓
Zero-shot	OPT-13B	0.00	100	0.10	100	0.05	100	3.38	100
	AdaInfer *w.* GAP	0.00	90.61	0.10	92.03	0.05	91.32	3.64	87.55
	AdaInfer *w.* CRF	0.00	97.55	0.10	97.55	0.05	97.55	4.11	97.55
	AdaInfer	0.00	96.87	0.10	100	0.05	98.44	3.86	92.52
Few-shot	OPT-13B	92.58	100	72.83	100	82.71	100	58.48	100
	AdaInfer *w.*GAP	94.20	78.30	12.95	82.54	53.58	80.42	48.20	85.50
	AdaInfer *w.* CRF	92.88	97.50	71.27	97.55	82.08	97.53	55.33	97.50
	AdaInfer	92.97	80.28	72.83	100	82.90	90.14	52.83	89.74
Zero-shot	Llama2-7B	0.00	100	0.10	100	0.05	100	5.47	100
	AdaInfer *w.*GAP	0.00	96.08	0.10	91.05	0.05	93.57	5.41	91.20
	AdaInfer *w.* CRF	0.00	96.07	0.10	92.20	0.05	94.14	3.62	92.08
	AdaInfer	0.00	96.37	0.10	91.36	0.05	93.87	5.32	91.55
Few-shot	Llama-2-7B	95.20	100	79.65	100	87.43	100	66.80	100
	AdaInfer *w.* GAP	95.30	67.78	79.72	94.38	87.51	81.08	66.80	87.99
	AdaInfer *w.* CRF	94.90	69.91	61.62	96.38	78.26	83.15	62.36	89.60
	AdaInfer	95.30	68.05	79.72	94.51	87.51	81.28	66.92	88.41
Zero-shot	Llama2-13B	0.00	100	0.10	100	0.05	100	2.32	100
	AdaInfer *w.* GAP	0.00	88.25	0.10	77.82	0.05	83.04	9.9	74.80
	AdaInfer *w.* CRF	0.00	97.27	0.10	94.04	0.05	95.66	3.43	90.29
	AdaInfer	0.00	97.43	0.10	88.37	0.05	92.90	6.14	85.76
Few-shot	Llama-2-13B	95.90	100	77.53	100	86.72	100	69.36	100
	AdaInfer *w.* GAP	91.45	51.25	69.17	70.65	80.31	60.95	53.78	70.38
	AdaInfer *w.* CRF	95.60	73.07	76.77	93.08	86.19	83.08	65.82	90.29
	AdaInfer	92.65	59.70	76.43	87.69	84.54	73.70	61.87	80.61