# Not all Layers of LLMs are Necessary during Inference

Anonymous ACL submission

## Abstract

The inference phase of Large Language Models (LLMs) is very expensive. An ideal inference stage of LLMs could utilize fewer computational resources while still maintaining its capabilities (*e.g.,* 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 instances; and deep layers for hard ones?" To answer this question, we first indicate that *Not all Layers are Necessary during Inference* by statistically analyzing the activated layers across tasks. Then, we propose a simple algorithm named AdaInfer to determine the inference termination moment based on the input instance adaptively. More importantly, AdaInfer does not alter LLM parameters and maintains generalizability across tasks. Experiments on well-known LLMs (*i.e.,* Llama2 series and OPT) show that AdaInfer can achieve an average of 17.8% pruning ratio, even up to 43% on sentiment tasks while maintaining comparable performance with minimal loss (<1%). Additionally, this method is orthogonal to other model acceleration techniques, potentially boosting inference efficiency further.

## 1 Introduction

LLMs have demonstrated impressive performance on various downstream tasks (*e.g.,* text generation, question & answering, and sentiment analysis) using various 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, 2022). Notably, In-context learning ability allows LLMs to adapt to tasks using input-output examples without parameter updates (Kossen et al., 2023; Todd et al., 2024). However, their inference phases are very expensive (Pope et al., 2023; Liu et al., 2023). For example, the inference time complexity for typical large models with Transformer structure is $LSd(d+S)$ per single inference, where $d$, $S$, and $L$ represent the hidden size, sequence length, and layer number, respectively. 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 popular methods for achieving efficient inference in LLMs include model pruning (Kim et al., 2024; Yang et al., 2024; Song et al., 2024; Men et al., 2024) and sparse models (LeCun et al., 1989; Liu et al., 2023). However, altering LLM parameters may risk compromising its generalization ability, which is challenging to detect. Meanwhile, different LLM designs pose compatibility challenges with other acceleration methods.

In this paper, we consider dynamically reducing the number of activated neurons as an approach to accelerate LLM inference. Inspired by the human thinking process (Salthouse, 1996; Deary et al., 2001), where quick answers are often provided for simple questions while more time is spent on thoughtful reasoning for complex ones, *e.g.,* knowledge-related questions. Previous studies (Teerapittayanon et al., 2016; Huang et al., 2017) show that "Easy" tasks activate at shallower layers while "hard" ones at deeper layers. Additionally, growth strategy (Li et al., 2023) is proposed to lower the training cost of LLMs by adding parameters in stages. It inspires us that reducing the computing parameters during inference may be an effective way besides existing typical accumulation methods. Statistical LLMs results on various tasks (see Section 3.2 for detail) show that reducing parameters is feasible during LLM inference.

Therefore, a natural approach to achieve LLM efficient inference is to decide when to stop the inference process based on the input instance adaptively. For instance, allocating fewer computational resources for processing "simple" samples to enhance operational efficiency. Furthermore, exploring adaptive inference may bridge LLMs with the brain's information processing (Hubel and Wiesel,

*Corresponding authors.

1962; Murata et al., 2000), aiding in the analysis of activated network modules during sample processing (Han et al., 2021) and identifying crucial input components affecting the final prediction.

Specifically, we present AdaInfer, a simple but effective algorithm for instance-aware adaptive inference. The core of AdaInfer lies in data-driven decision-making. Generally, there are two approaches to getting decision-making signals: (1) updating LLM parameters requires training, involves high costs, and might decrease the model's generalizability (Gu et al., 2024), and (2) keeping parameters unchanged, a more desirable and cost-effective approach that preserves the model's innate ability (Yao et al., 2023). In this work, we adopt an *Early Exit (EE)* strategy, optimizing efficiency without altering the model's parameters. *EE* was utilized in accelerating the inference of encoder-only architectures in BERT by (Li et al., 2021; Kong et al., 2022). Our proposed AdaInfer closely aligns with this *EE* concept. In particular, we begin 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 features and employ classical statistical classifiers (*i.e.,* SVM and CRF) to facilitate the early exit strategy (Section 4).

Experiments on well-known LLMs (*i.e.,* Llama2 series and OPT) show that AdaInfer can achieve an average of 17.8% pruning ratio, even up to 43% on sentiment tasks while maintaining comparable performance with minimal loss (<1% ). More importantly, AdaInfer is orthogonal to other model acceleration techniques, offering the potential for further enhancing inference efficiency (Section 5).

## 2 Related Work

**Adaptive Inference.** A straightforward approach to achieve adaptive inference involves dynamic depth neural networks (Han et al., 2021; Huang et al., 2017; Bolukbasi et al., 2017). Dynamic depth involves two methods: *Early Exit (EE)* and *Skip layer*. *EE* first appeared in CNN/DNN networks for visual tasks (Bolukbasi et al., 2017; Huang et al., 2017; Teerapittayanon et al., 2016). Subsequently, it was utilized in accelerating the inference of encoder-only architectures in BERT by (Li et al., 2020; Liu et al., 2020; Li et al., 2021; Kong et al., 2022). Recently, (Schuster et al., 2022; Varshney et al., 2023) discuss confidence-based *EE* for LM adaptive inference. Our proposed AdaInfer closely aligns with *EE* concept. Specifically, we apply *EE* to mainstream decoder-only LLMs, which adhere to the scaling law but suffer from high inference costs due to their large parameter count. Meanwhile, skip-layer dynamically omits the execution of middle layers (or modules) for any input token, facilitated by a gate function (Wang et al., 2018) or a binary router (Zeng et al., 2023; Raposo et al., 2024). The main difference between our method and theirs is that we achieve instance-wise inference without altering the model parameters, which is crucial for current LLMs. To the best of our knowledge, this is the first attempt to discover that each block's logits are crucial elements for *EE* classifiers in LLMs, and we incorporate it as a fundamental design choice in AdaInfer.

**Model Compression.** Techniques like Quantization (Xiao et al., 2023; Xing et al., 2023), Sparsity (Liu et al., 2023; Frantar and Alistarh, 2023), Distillation (Touvron et al., 2021; Tang et al., 2019) has been developed to improve the inference efficiency of LLMs. Another line of work in model compression is network pruning (Kim et al., 2024; Yang et al., 2024; Song et al., 2024; Men et al., 2024; Ma et al., 2023). Depth pruning (Ma et al., 2023) is often considered less effective in performance compared to width pruning (Xia et al., 2024) because it involves eliminating larger and more coarse units (Kim et al., 2024). Our method aligns with depth pruning techniques and achieves comparable performance to dense implementations. The key distinction of our approach is its dynamic pruning ratio tailored to specific tasks.

## 3 Efficiency Analysis of LLM Inference

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

### 3.1 Preliminary: LLM Building Blocks

Modern LLMs, rooted in the Transformer architecture (Vaswani et al., 2017), are trained with different unsupervised training objectives. For instance, mainstream LLMs (*e.g.,* GPT, Llama series) are pre-trained 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:

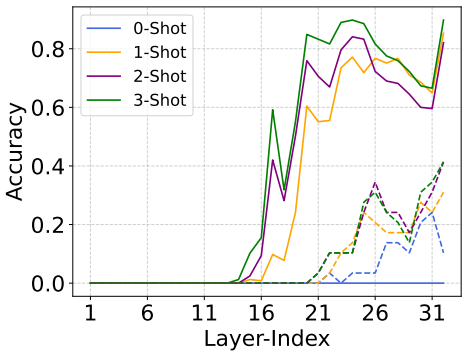



Figure 1: LLama2-7B model zero/few-shot performance across all decoder layers: solid line for sentiment analysis while dashed line for MMLU tasks.

**Tokenizer and Embedding Layer.** This block tokenizes input text into numerical vectors, enabling effective processing and analysis of textual data.

**Decoder Block.** This block processes numerical vectors through self-attention and feedforward neural networks, enabling the model to focus on (attend to) the most relevant input parts.

**Classification Layer.** The LM head layer converts decoder logits into a vocabulary-wide probability distribution to facilitate word prediction.

These blocks facilitate LLMs in efficiently handling NLP downstream tasks, with a primary emphasis on decoder blocks within multi-layer Transformers. For typical large Transformer models, inference complexity is linearly related to the number of decoder layers $L$ and is given by $LSd(d+S)$ per single inference. Consequently, to explore the possibility of skipping intermediate layers in LLMs during inference, we do the following statistics.

### 3.2 Not all Layers are Necessary

Earlier Transformer models typically comprise 6 decoder layers, while current open-source models, such as Llama2-13B (Touvron et al., 2023), 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 conduct a statistical analysis to examine the correlation between accuracy and the activation of layers across various tasks. The statistical results are depicted in Figure 1.

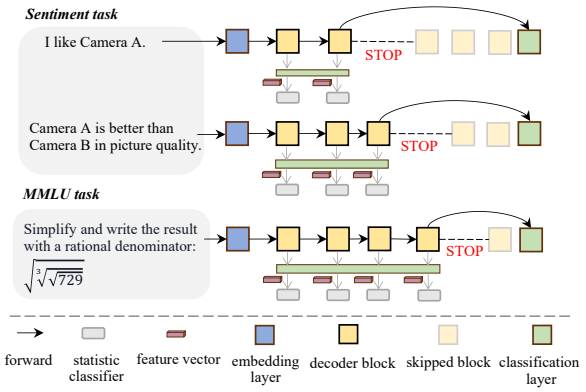
**Observation 1:** *Not all Layers of LLMs are Necessary during Inference: Early Stopping works.* In sentiment analysis using the Llama2-13B (40 layers) model, the average activated layer count per input is 21, with a variance of 5.1. This observation is intuitive. For instance, simpler inputs like "I 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.

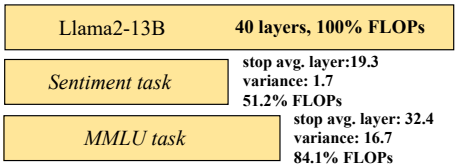
**Insight.** The observations mentioned above are intuitive. By exploiting this phenomenon, we can perform instance-aware adaptive inference for LLMs, dynamically adjusting their structure/parameters for different test samples, thereby achieving superior advantages in inference efficiency and adaptability. Moving forward, we will leverage this observation to implement adaptive inference.

## 4 AdaInfer

The workflow of AdaInfer and the computational efficiencies gained through this method are depicted in Figures 2a and 2b, respectively. The key of AdaInfer is how to find the early stop signal while keeping the original abilities of LLMs. AdaInfer dynamically computes the stopping signal by evaluating critical features (*i.e.,* "gap" and "top prob"). This process involves two main components: a Feature Selection module and a Classifier module. At each layer, the Feature Selection crafts a feature vector for the current input instance. Subsequently, the Classifier (often SVM or CRF for their effectiveness) assesses the stopping signal's




(a) A workflow of AdaInfer processing three input instances, involving two for sentiment analysis and one for a knowledge-based question answering task. It shows that the early-exit moment varies across the instances.




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

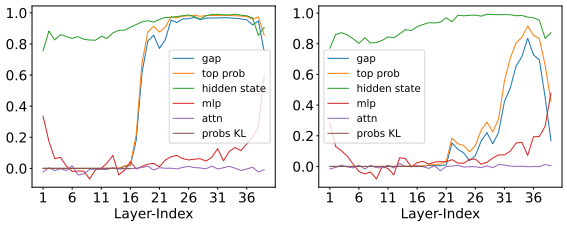
Figure 2: An illustration of AdaInfer's processing and computational savings.

strength. A strong enough signal triggers an early process termination, allowing for the bypass of subsequent decoder layers.

### 4.1 Feature Selection

As we mentioned before, modifying LLM parameters requires training and incurs high costs. More importantly, it may pose a potential risk of compromising the model's generalization capabilities and detecting these issues can be challenging (Gu et al., 2024). Hence, we embrace a more desirable and cost-effective approach that preserves the model's innate abilities without altering parameters. AdaInfer utilizes specially designed features (*e.g.,* "gap" and "top prob"), leveraging a statistical classifier for evaluation stopping signal.

**Problem: The lack of features for decision-making.** 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.




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

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

**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 with 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, OPT-13B (Zhang et al., 2022), and GPT-J (Wang and Komatsuzaki, 2021) (See Appendix C). This demonstrates **"gap" and "top prob" can serve as universal features**, indicating the stopping signal. Notably, these two values remain consistent across diverse tasks, suggesting a versatile classifier applicable to various tasks. Factor study in subsequent experiments also shows that other features (*e.g.,* Cosine Similarity) exhibit subtle differences across layers.

### 4.2 Classifier

The Classifier determines if the signal is compelling enough to warrant an early termination of the process. The rule-based approach heavily relies on rules, and the cost of individually constructing domain-specific features is high (Huang et al., 2017; Yang et al., 2020; Wang et al., 2022). 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.

The trend in Figure 3 indicates classical statistical classification methods can address discrete decision-making problems. We can connect block features to decision-making via a statistical classifier. By classifying general features (*i.e.,* "gap" and "top prob"), we simplify decision-making into binary classification, enabling an early exit strategy. If the classifier considers the current layer's features stoppable, subsequent layers' computations can be discarded; otherwise, continue to the final layer. This process is also illustrated in Figure 2a.

### 4.3 Classifier Objective

Here we detail the training process of the classifier through their objectives, respectively. 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 B. 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.

Table 1: LLMs statistics using AdaInfer.

| Model | Params | Tokens | Layer Num. |
|---|---|---|---|
| Meta/OPT | 13B | 0.18T | 40 |
| Meta/Llama 2 | 7B | 2T | 32 |
| Meta/Llama 2 | 13B | 2T | 40 |
| Meta/Llama 2 | 70B | 2T | 80 |

**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.

## 5 Experiments

We now conduct experiments with AdaInfer on well-known LLMs across various tasks.

### 5.1 Evaluation Tasks

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

**Question Answering Tasks.** (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 either segments of passage or marked as unanswerable.

**Text Classification Tasks.** (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.

### 5.2 Experiment Settings

**Large Language Models.** For AdaInfer's backbone, we choose widely recognized LLMs, detailed in Table 1. These models vary in terms of the number of parameters, ranging from 7 billion to 70 billion, and the number of layers, ranging from 32 layers to 80 layers. Specifically, our selections encompass OPT (Zhang et al., 2022) and the Llama 2 series (Touvron et al., 2023). These models exhibit subtle differences in architectural design and

Table 2: Performance and computational efficiency in multi-tasks, with accuracy (%) denoted by 'Acc'. Results include few-shot learning with sample sizes of 5, 10, 15, and 20, showcasing the average values.

| Tasks | P. Ratio(%) | MMLU | | | CommonsenseQA | | | SQuAD | | | Sentiment | | | AG News | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Acc | #Avg. L | Var | Acc | #Avg. L | Var | Acc | #Avg. L | Var | Acc | #Avg. L | Var | Acc | #Avg. L | Var |
| ***Llama 7B*** | | *Total 32 layers* | | | | | | | | | | | | | | |
| Dense | 0.00 | 43.05 | 32.00 | 0.00 | 53.50 | 32.00 | 0.00 | 20.40 | 32.00 | 0.00 | 95.20 | 32.00 | 0.00 | 79.65 | 32.00 | 0.00 |
| ShortGPT | 28.13 | 21.52 | 23.00 | 0.00 | 33.52 | 23.00 | 0.00 | 10.60 | 23.00 | 0.00 | 93.48 | 23.00 | 0.00 | 56.90 | 23.00 | 0.00 |
| ShortGPT | 15.63 | 29.95 | 27.00 | 0.00 | 41.90 | 27.00 | 0.00 | 12.97 | 27.00 | 0.00 | 90.4 | 27.00 | 0.00 | 53.25 | 27.00 | 0.00 |
| ShortGPT | 9.38 | 37.39 | 29.00 | 0.00 | 53.22 | 29.00 | 0.00 | 14.32 | 29.00 | 0.00 | 94.17 | 29.00 | 0.00 | 71.28 | 29.00 | 0.00 |
| **AdaInfer** | 9.66 → 35.71 | 43.73 | 28.91 | 4.97 | 53.00 | 27.90 | 5.93 | 45.82 | 26.77 | 11.88 | 95.30 | 20.57 | 5.10 | 79.72 | 29.20 | 2.70 |
| ***Llama 13B*** | | *Total 40 layers* | | | | | | | | | | | | | | |
| Dense | 0.00 | 53.31 | 40.00 | 0.00 | 64.92 | 40.00 | 0.00 | 52.90 | 40.00 | 0.00 | 95.90 | 40.00 | 0.00 | 77.53 | 40.00 | 0.00 |
| ShortGPT | 25.00 | 45.12 | 30.00 | 0.00 | 65.00 | 30.00 | 0.00 | 13.32 | 30.00 | 0.00 | 84.38 | 30.00 | 0.00 | 55.90 | 30.00 | 0.00 |
| ShortGPT | 12.50 | 46.64 | 35.00 | 0.00 | 64.45 | 35.00 | 0.00 | 16.35 | 35.00 | 0.00 | 89.8 | 35.00 | 0.00 | 70.17 | 35.00 | 0.00 |
| ShortGPT | 7.50 | 47.22 | 37.00 | 0.00 | 64.47 | 37.00 | 0.00 | 17.25 | 37.00 | 0.00 | 95.90 | 37.00 | 0.00 | 75.47 | 37.00 | 0.00 |
| **AdaInfer** | 9.13 → 43.33 | 52.44 | 36.35 | 8.15 | 62.48 | 34.60 | 10.20 | 48.35 | 31.18 | 31.75 | 92.65 | 22.67 | 8.10 | 76.43 | 34.02 | 24.18 |
| ***OPT 13B*** | | *Total 40 layers* | | | | | | | | | | | | | | |
| Dense | 0.00 | 23.60 | 40.00 | 0.00 | 21.45 | 40.00 | 0.00 | 26.12 | 40.00 | 0.00 | 92.58 | 40.00 | 0.00 | 72.83 | 40.00 | 0.00 |
| ShortGPT | 25.00 | 10.17 | 30.00 | 0.00 | 11.5 | 30.00 | 0.00 | 0.65 | 30.00 | 0.00 | 14.72 | 30.00 | 0.00 | 2.27 | 30.00 | 0.00 |
| ShortGPT | 12.50 | 22.92 | 35.00 | 0.00 | 19.12 | 35.00 | 0.00 | 22.12 | 35.00 | 0.00 | 86.33 | 35.00 | 0.00 | 49.42 | 35.00 | 0.00 |
| ShortGPT | 7.50 | 23.05 | 37.00 | 0.00 | 19.68 | 37.00 | 0.00 | 24.65 | 37.00 | 0.00 | 91.35 | 37.00 | 0.00 | 66.62 | 37.00 | 0.00 |
| **AdaInfer** | 9.75 → 22.63 | 22.59 | 32.37 | 7.92 | 21.62 | 33.33 | 12.12 | 25.95 | 34.20 | 13.50 | 92.97 | 30.95 | 5.77 | 72.83 | 39.00 | 0.00 |

training data volume.

**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: $\mathrm{Q}:\{x_k\}\backslash\mathrm{nA}:\{y_k\}\backslash\mathrm{n}\backslash\mathrm{n}$, concatenating random $x_k$ and $y_k$ samples from task-specific training sets.

**Metrics.** For performance evaluation, we report the top-1 accuracy score on the test set following function vectors (Todd et al., 2024) (HELM implementation)[1]. To evaluate computational efficiency, we follow previous works (Ma et al., 2023; Schuster et al., 2022; Elbayad et al., 2019) and report the pruning ratio (P. Ratio) and the average number of activated layers (#Avg. L) for each task, along with their variance (Var). These metrics directly measures complexity reduction, avoiding conflation with implementation or infrastructure-specific details (Dehghani et al., 2021). For reference, we also translated them into FLOPs reduction ratios in the Appendix D. Considering the conditional checks and classifier computation involved in AdaInfer, we also compared the actual speed of AdaInfer in real-world scenarios with Dense implementation, reporting wall-clock time (Dehghani et al., 2021).

**Comparison methods.** We compared our method with the structured pruning method ShortGPT (Men et al., 2024), which prunes redundant layers in LLMs based on similarity scores. For the OPT model, we calculated redundant layers as outlined in the paper. For the LLama model, we used the same layers reported. Note that these model pruning methods apply a static pruning ratio across all tasks, whereas our AdaInfer adaptively performs model pruning based on input.

[1]https://huggingface.co/blog/open-llm-leaderboard-mmlu

### 5.3 Main Results

The main results of AdaInfer are presented in Table 2. Conducted in few-shot settings, these experiments show the Top-1 accuracy, pruning ratios, average active layers for each task, and their variance compared to the baseline. From the perspective of performance and computational efficiency, we can draw the following experimental conclusions.

**Performance is Comparable with Minimal Loss (<1%).** Tables 2 shows that Top-1 accuracy remains within a narrow margin of $<1\%$ for all tasks when compared to dense models. Since AdaInfer adaptively reduces model parameters, with the pruning ratio determined based on the task, we conducted experiments on ShortGPT (Men et al., 2024) to match the adaptive pruning range of AdaInfer. This allows for a more comprehensive comparison with methods that use a fixed pruning ratio (Yang et al., 2024; Ma et al., 2023). The results are shown in Table 2, demonstrating that AdaInfer surpasses the baseline method with stable performance. It achieves adaptive inference while maintaining LLM capabilities and in-context learning

Table 3: Wall-clock time (s) and actual speedup for 358 test samples from MMLU and 245 test samples from Sentiment Tasks.

| Task | Llama2 7B (FP32) | | | Llama2 13B (FP16) | | |
|---|---|---|---|---|---|---|
| | Dense | **Ours** | Speed up | Dense | **Ours** | Speed up |
| MMLU | 796.53 | 781.31 | 1.02x | 339.19 | 320.46 | 1.05x |
| Sentiment | 41.18 | 39.69 | 1.04x | 28.18 | 21.76 | 1.30x |

abilities [2] without modifying model parameters.

This finding is promising, especially in light of our observation1 in Section 3.2, where we demonstrate the feasibility of implementing early exit strategies within LLM middle layers while preserving performance. For certain tasks, AdaInfer surpasses the last layer accuracy. This hints at a tendency for deep layers to potentially over-represent certain tasks, which could impede performance during LLM inference.

**Pruning ratio from 9% to 43%, average 17.8%.** We report the average and variance of activated layers for each task and convert them to pruning ratios in Table 2. It can be observed that the pruning ratios vary for different types of tasks, ranging from $9\%$ to $43\%$. This variation is because 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 $43\%$ 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 $9\%$ to $20\%$. This aligns with our observation2 outlined in Section 3.2, 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.

**Wall-clock time.** Next, we study the end-to-end runtime of AdaInfer. Table 3 compares the runtime of AdaInfer with a dense implementation on MMLU and Sentiment tasks (5-shot, batch size set to 1), using $6 \times V100$ (32GB) hardware. We observed a 1.03x speed up on MMLU and 1.17x speed up on Sentiment when applying AdaInfer. This indicates that despite AdaInfer converting hidden states to logits at each block through the LM head layer, it only utilizes the last token's hidden state even with longer sequences. Consequently, this computation is minimal (0.03% of the total FLOPs for transformer inference). Further computational details on this process can be found in Appendix A. Meanwhile, statistical classifiers (e.g., SVM) have significantly lower computational costs compared to LLM inference, as detailed in Appendix A, highlighting the computational efficiency potential of AdaInfer.

[2]We noted a decline in the performance of the reproduced ShortGPT on the SQuAD dataset when the prompts increased to 10, 15, and 20 shots.

Table 4: Comparative analysis of GAP and CRF on performance and computational efficiency.

| Task | Setting | AdaInfer *w.* Rule | | 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 |

### 5.4 Evaluation on Different Exit Strategy

In the main experiments Table 2 , we employ 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 from 3% to 50% and maintain comparable LLM performance. Notably, in the zero-shot setting, GAP outperforms CRF, suggesting a relatively weak dependency between block features.

### 5.5 Evaluation across Scaling Law

In our main experiments in Table 2, we employ 7B/13B sized Llama2 and OPT models. In experiments with the Llama2 70B version, we observe that in a zero-shot setting, AdaInfer matches or slightly exceeds the baseline model while reducing computational costs by 10% to 50%. However, in the few-shot setting, despite similar reductions in computation, AdaInfer's accuracy shows a 1% to 25% drop 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. The results of all LLMs using different classifiers are summarized in Table 7 and Table 8 in the Appendix D and we have highlighted the best results for each task in the current setting.

### 5.6 Generalization Study

In Tables 2, we randomly select 6 training datasets from the entire pool of task training sets, which altogether contain 71 sub-datasets, to train the AdaInfer classifier. Furthermore, to assess the generalization performance of the statistical classifiers, we conduct 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 5. 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 5, when using AdaInfer, we recommend utilizing SVM as the classifier.

### 5.7 Factor Study

In response to the features mentioned in Section 4.1, we conduct cross-validation. Given that the classifiers in the main results utilized basic features (*i.e.,* "gap", "top prob"), we explore 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 results are presented in Table 6. The attention values have 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 other features may involve various factors.

Table 5: Generalization performance of statistic classifier on sentiment task on Llama2-7B (32 layers), Inter-Model refers to Llama2-13B (40 layers).

| Classifier | Generalization | Acc | Layers | Variance | FLOPs |
|---|---|---|---|---|---|
| SVM | Intra-Task | 94.90 | 18.15 | 0.45 | 60.58 |
| CRF | | 0.00 | 0.00 | 0.00 | 0.00 |
| SVM | Inter-Task | 95.50 | 19.20 | 4.40 | 63.80 |
| CRF | | 94.90 | 20.20 | 4.55 | 66.87 |
| SVM | Inter-Model | 90.70 | 20.60 | 3.70 | 54.55 |
| CRF | | 87.75 | 19.20 | 2.75 | 51.09 |

Table 6: 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 |

## 6 Conclusion

In this paper, we first give evidence of that *Not all Layers are Necessary during Inference* and provide statistical evidence to support this. Then, we present AdaInfer, a simple yet effective algorithm that 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 well-known LLMs (*i.e.,* Llama2 series and OPT) show that AdaInfer can achieve an average of 17.8% pruning ratio, even up to 43% on sentiment tasks while maintaining comparable performance with minimal loss (<1% ). More importantly, AdaInfer is compatible with other model acceleration techniques, potentially offering further improvements in inference efficiency. We argue that AdaInfer establishes a new paradigm for efficient inference besides effective existing methods.

## Limitations

In this paper, we make the 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


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.

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.

Mostafa Dehghani, Anurag Arnab, Lucas Beyer, Ashish Vaswani, and Yi Tay. 2021. The efficiency misnomer. arXiv preprint arXiv:2110.12894.

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

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

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).

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.

Bo-Kyeong Kim, Geonmin Kim, Tae-Ho Kim, Thibault Castells, Shinkook Choi, Junho Shin, and Hyoung-Kyu Song. 2024. Shortened llama: A simple depth pruning for large language models. arXiv preprint arXiv:2402.02834.

Jun Kong, Jin Wang, Liang-Chih Yu, and Xuejie Zhang. 2022. Accelerating inference for pretrained language models by unified multi-perspective early exiting. In Proceedings of the 29th International Conference on Computational Linguistics, pages 4677–4686.

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.

Lei Li, Yankai Lin, Deli Chen, Shuhuai Ren, Peng Li, Jie Zhou, and Xu Sun. 2020. Cascadebert: Accelerating inference of pre-trained language models via calibrated complete models cascade. arXiv preprint arXiv:2012.14682.

Xiang Li, Yiqun Yao, Xin Jiang, Xuezhi Fang, Xuying Meng, Siqi Fan, Peng Han, Jing Li, Li Du, Bowen Qin, Zheng Zhang, Aixin Sun, and Yequan Wang. 2023. FLM-101B: an open LLM and how to train it with $100k budget. CoRR, abs/2309.03852.

Xiaonan Li, Yunfan Shao, Tianxiang Sun, Hang Yan, Xipeng Qiu, and Xuanjing Huang. 2021. Accelerating bert inference for sequence labeling via early-exit. arXiv preprint arXiv:2105.13878.

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

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

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.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. Advances in neural information processing systems, 36:21702–21720.

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.

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.

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.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. 2024. Mixture-of-depths: Dynamically allocating compute in transformer-based language models.

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

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. 2022. Confident adaptive language modeling. Advances in Neural Information Processing Systems, 35:17456–17472.

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.

Jiwon Song, Kyungseok Oh, Taesu Kim, Hyungjun Kim, Yulhwa Kim, and Jae-Joon Kim. 2024. Sleb: Streamlining llms through redundancy verification and elimination of transformer blocks. arXiv preprint arXiv:2402.09025.

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.

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.

Neeraj Varshney, Agneet Chatterjee, Mihir Parmar, and Chitta Baral. 2023. Accelerating llama inference by enabling intermediate layer decoding via instruction tuning with lite. arXiv e-prints, pages arXiv–2310.

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.

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.

Yequan Wang, Hengran Zhang, Aixin Sun, and Xuying Meng. 2022. CORT: A new baseline for comparative opinion classification by dual prompts. In Findings of the Association for Computational Linguistics: EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022, pages 7064–7075.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2024. Sheared llama: Accelerating language model pre-training via structured pruning.

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.

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

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.

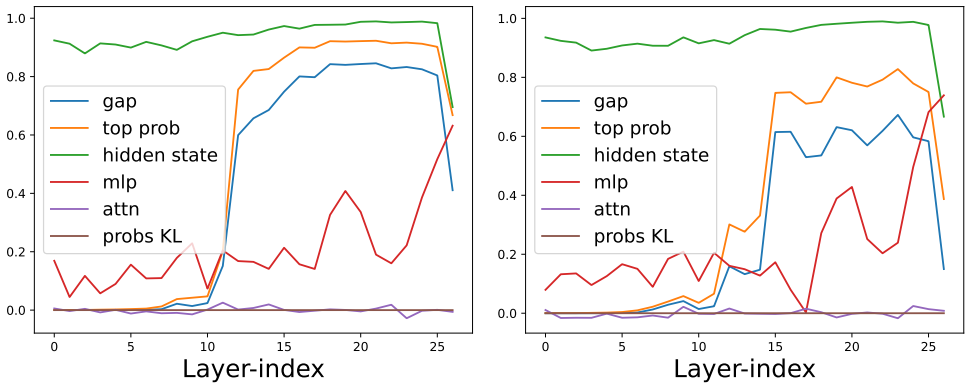
Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. Advances in neural information processing systems, 28.

## A Computation Cost.

**Classifier Computation Cost.** We utilized the sklearn library for training SVM[3] and CRF[4], 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 hidden size, $S$ 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. While AdaInfer does convert hidden states to logits at each block through classification layer, it only utilizes the hidden state from the last token for conversion, even when the sequence length is 2048 or longer. In the case of Llama2 7/13/70B, this computation accounts for only 0.000288, 0.000236, and 0.000152 of the total number of FLOPs for transformer inference. Similarly, for OPT 13B, it amounts to 0.000367. Consequently, the computational burden associated with this aspect can be disregarded. Summing these together, a transformer model with $l$ transformer layers, the total number of floating-point operations for inference

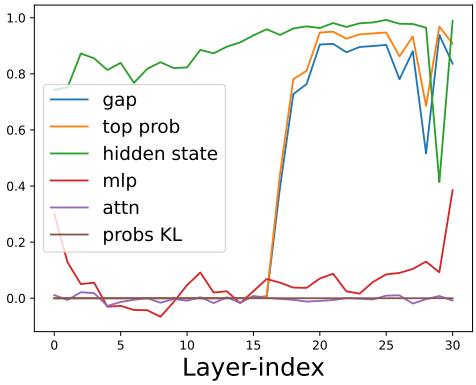
[3]https://scikit-learn.org/stable/modules/svm.html
[4]https://sklearn-crfsuite.readthedocs.io/en/latest/



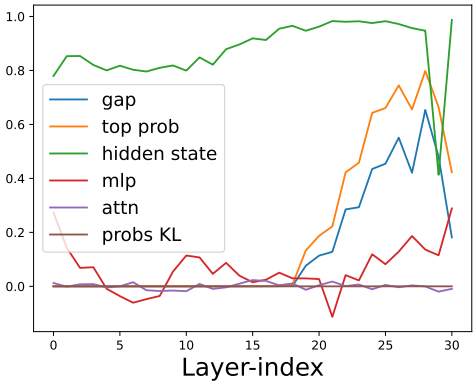

(a) GPT-J 6B on sentiment



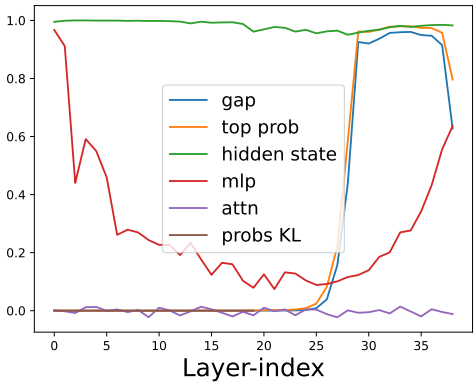

(b) GPT-J 6B on MMLU



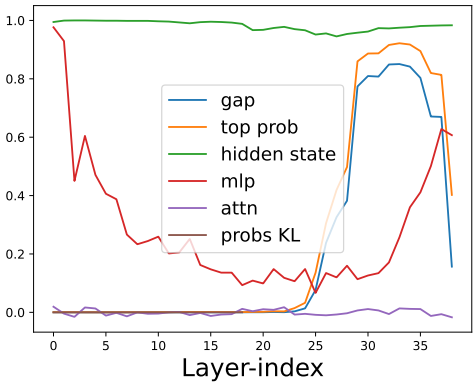

(c) Llama2-7B on sentiment




(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.

is $4Bshl(6h + s) + 2BshV$. Thus, the ratio of inference cost in FLOPs can be calculated as

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

## B 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 3.1). This can be represented as: $P = \mathrm{softmax}(WH + b)$, where $H$ is the hidden state of the current layer, $W$ and $b$ are the weights and bias of the classification layer, respectively. Function $\mathrm{softmax}$ is applied to convert logits to probabilities. Let the highest-ranked token in this distribution be denoted as $\hat{y} = \mathrm{argmax}(P)$, where $\mathrm{argmax}(P)$ 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. Thus, for an $L-$layer LLM, each input instance $x$ yields $L$ pairs of $< x^d, y_c >$.

## C More Observation of LLMs

Figure 4 depicts a visual analysis of features across the layers within each block of mainstream LLMs. It shows that the "gap" and "top prob" exhibit a gradual increase during the inference phase, reaching stability in the deeper layers. Additionally, the activation of "gap" and "top prob" varies across layers for different tasks. These observed trends align with the findings discussed in Section 4.1.

## D Comprehensive Summary of Results

The results of all LLMs using different classifiers are summarized in Table 7 and 8. We have highlighted the best results for each task in the current setting. The experimental results indicate that (i) early exits are feasible for different tasks, (ii) the timing of early exits varies depending on the instance, and (iii) in both zero-shot and few-shot settings, accuracy is comparable with baseline models. It's worth noting that for individual tasks, AdaInfer even outperforms the baseline in zero-shot or few-shot accuracy. This suggests that in inference scenarios, deep layers may tend to over-represent some tasks, potentially impairing performance.

Table 7: 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.* Rule | 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.* Rule | 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.* Rule | 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.* Rule | 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.* Rule | 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.* Rule | 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 8: 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.* Rule | 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.* Rule | 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.* Rule | 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.* Rule | 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.* Rule | 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.* Rule | 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 |